Wie arbeitet ein Mikrocontroller?

Anwenderhandbuch

VERSION 2.2 GER

DEUTSCHE ÜBERSETZUNG: REINHARD K. RAHNER

PARALLAX INC

GARANTIE

Parallax Inc. gibt auf seine Produkte bei Materialschäden eine Herstellergarantie von 90 Tagen ab Kaufdatum. Wird ein Fehler festgestellt, behält sich Parallax Inc. vor, den Schaden zu reparieren, die Ware auszutauschen oder den Kaufpreis zurück zu erstatten. Ein Rückversand an Parallax kann nur mit einer vorher zugewiesenen Return Merchandise Authorization (RMA) Nummer erfolgen. Die RMA Nummer muss deutlich sichtbar auf das Rücksendepaket geschrieben werden. Dem Rücksendepaket sind die folgenden Informationen beizufügen: Name des Absenders, Telefonnummer, Versandadresse und eine Beschreibung des aufgetretenen Fehlers. Parallax wird das Gerät oder ein Austauschgerät auf dem gleichen Postweg zurückschicken wie es versandt wurde.

14 TAGE GELD ZURÜCK GARANTIE

Wenn Sie innerhalb von 14 Tagen nach Kauf dieses Produktes feststellen, dass es Ihren Anforderungen nicht genügt, können Sie es an uns zurückschicken. Parallax Inc. zahlt den vollen Kaufpreis abzüglich von Versand- und Bearbeitungskosten zurück. Die Garantie erlischt, wenn das Produkt verändert oder beschädigt wurde. Weitere Hinweise für das Zurückschicken eines Produktes an Parallax finden Sie weiter oben im Abschnitt Garantie.

COPYRIGHTS AND TRADEMARKS

This documentation is copyright 2003-2004 by Parallax Inc. By downloading or obtaining a printed copy of this documentation or software you agree that it is to be used exclusively with Parallax products. Any other uses are not permitted and may represent a violation of Parallax copyrights, legally punishable according to Federal copyright or intellectual property laws. Any duplication of this documentation for commercial uses is expressly prohibited by Parallax Inc. Duplication for educational use is permitted, subject to the following Conditions of Duplication: Parallax Inc. grants the user a conditional right to download, duplicate, and distribute this text without Parallax's permission. This right is based on the following conditions: the text, or any portion thereof, may not be duplicated for commercial use; it may be duplicated only for educational purposes when used solely in conjunction with Parallax products, and the user may recover from the student only the cost of duplication.

This text is available in printed format from Parallax Inc. Because we print the text in volume, the consumer price is often less than typical retail duplication charges.

BASIC Stamp, Stamps in Class, Board of Education and SumoBot are registered trademarks of Parallax, Inc. HomeWork Board, Boe-Bot and Toddler are trademarks of Parallax Inc. If you decide to use the words BASIC Stamp, Stamps in Class, Board of Education, HomeWork Board, Boe-Bot or Toddler on your web page or in printed material, you must state that "BASIC Stamp is a registered trademark of Parallax Inc.", "Stamps in Class is a registered trademark of Parallax Inc.", "Board of Education is a registered trademark of Parallax Inc.", "SumoBot is a registered trademark of Parallax Inc." "HomeWork Board is a trademark of Parallax Inc.", "Boe-Bot is a trademark of Parallax Inc.", or "Toddler is a trademark of Parallax Inc." respectively, upon the first appearance of the trademark name. Other brand and product names are trademarks or registered trademarks of their respective holders.

ISBN 1-928982-02-6

DISCLAIMER OF LIABILITY

Parallax Inc. is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of

equipment or property, or any costs of recovering, reprogramming, or reproducing any data stored in or used with Parallax products. Parallax Inc. is also not responsible for any personal damage, including that to life and health, resulting from use of any of our products. You take full responsibility for your BASIC Stamp application, no matter how life-threatening it may be.

WEB SITE AND DISCUSSION LISTS

The Parallax Inc. web site (www.parallax.com) has many downloads, products, customer applications and on-line ordering for the components used in this text. We also maintain several e-mail discussion lists for people interested in using Parallax products. These lists are accessible from www.parallax.com via the Support → Discussion Groups menu. These are the lists that we operate:

- BASIC Stamps – With over 4,000 subscribers, this list is widely utilized by engineers, hobbyists and students who share their BASIC Stamp projects and ask questions.
- Stamps in Class – Created for educators *and* students, this list has about 650 subscribers who discuss the use of the Stamps in Class curriculum in their courses. The list provides an opportunity for both students and educators to ask questions and get answers.
- Parallax Educators –Exclusively for educators and those who contribute to the development of Stamps in Class. Parallax created this group to obtain feedback on our curricula and to provide a forum for educators to develop and obtain Teacher's Guides.
- Parallax Translators – Consisting of about 40 people, the purpose of this list is to provide a conduit between Parallax and those who translate our documentation to languages other than English. Parallax provides editable Word documents to our translating partners and attempts to time the translations to coordinate with our publications.
- Toddler Robot – A customer created this discussion list to discuss applications and programming of the Parallax Toddler robot.
- SX Tech – Discussion of programming the SX microcontroller with Parallax assembly language tools and 3rd party BASIC and C compilers. Approximately 600 members.
- Javelin Stamp – Discussion of application and design using the Javelin Stamp, a Parallax module that is programmed using a subset of Sun Microsystems' Java® programming language. Approximately 250 members.

DRUCKFEHLER

Obwohl wir uns große Mühe bei der Erstellung dieses Textes gegeben haben, können sich doch Fehler eingeschlichen haben. Sollten Sie einen Fehler finden, schicken Sie bitte eine email an editor@parallax.com. Wir sind bemüht unser Unterrichtsmaterial und die Dokumentationen ständig zu verbessern und dort wo Fehler festgestellt wurden, diese möglichst umgehend zu dokumentieren. Sollte dies der Fall sein, werden wir eine Korrekturliste auf unserer Website unter www.parallax.com hinterlegen.

HINWEIS DES ÜBERSETZERS

Die Informationen in diesem Buch werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht. Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt. Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Parallax Inc. und der Übersetzer können für fehlerhafte Angaben und deren Folgen weder

eine juristische Verantwortung noch irgendeine Haftung übernehmen. Für Verbesserungsvorschläge und Hinweise auf Fehler sind Parallax Inc. und der Autor dankbar.
Die Dokumentation unterliegt dem Urheberrecht. Alle Rechte vorbehalten. Kein Teil dieses Handbuches darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Genehmigung durch Parallax Inc. reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden. Hiervon sind die in den Paragraphen 53, 54 UrhG ausdrücklich genannten Ausnahmefälle nicht berührt.
Es wurden alle Anstrengungen unternommen, die Richtigkeit und Vollständigkeit der Angaben in diesem Handbuch zu gewährleisten. Trotzdem können Fehler existieren. Parallax Inc. und der Übersetzer können keine Verantwortung für Schäden irgendwelcher Art übernehmen, die durch die Benutzung von Informationen in diesem Handbuch oder die Nutzung eines Programmes auftreten können.

Reinhard K. Rahner
D-24214 Gettorf

Inhaltsverzeichnis

Vorwort 5
Hinweis an den Leser 5
Unterstützung und Diskussionsforen 5
Lehrerhandbuch 6
Der Stamps in Class Lehrgang 6
Übersetzungen 8
Danksagungen 8

Kapitel 1: Erste Schritte mit der BASIC Stamp 1
Mikrocontroller, unsere heimlichen Helfer 1
Die BASIC Stamp 2 1
Herausragende Projekte mit BASIC Stamp Mikrocontrollern 2
Hardware und Software 5
Übung 1: Der BASIC Stamp Editor 6
Übung 2: Installation der Software 10
Übung 3: Hardware einrichten und Systemtest 13
Übung 4: Ein erstes Programm 21
Übung 5: Hilfe anfordern 28
Übung 6: Der ASCII Code 31
Übung 7: Die Arbeit beenden 32
Zusammenfassung 35

Kapitel 2: Leuchtdioden ansteuern 39
Kontrollleuchten 39
Eine Leuchtdiode einschalten 39
Übung 1: Aufbau und Test einer LED Schaltung 40
Übung 2: Die BASIC Stamp als Ein/Aus Schalter 48
Übung 3: Zählen und Wiederholen 54
Übung 4: Aufbau und Test einer zweiten LED Schaltung 58
Übung 5: Ansteuerung einer DUO-LED 62
Zusammenfassung 70

Kapitel 3 : Digitale Eingabe mit Taster 75
Taschenrechner, Game-Boys und andere Geräte 75
Senden und Empfangen von HIGH/LOW Signalen 75
Übung 1: Tasterverhalten mit einer LED Schaltung prüfen 75
Übung 2: Tasterabfrage mit der BASIC Stamp 79
Übung 3: Tastersteuerung einer LED Schaltung 84
Übung 4: Zwei Taster steuern zwei LED Schaltungen 87
Übung 5: Bau eines Reaktionszeitmessers 93
Zusammenfassung 102

Kapitel 4: Servosteuerung 109
Bewegung mit Mikrocontrollern steuern 109
Motorbewegung und ON/OFF Signale 109
Übung 1: Aufbau und Test eines Servos 109
Übung 2: Positionssteuerung mit dem Computer 125
Übung 3: Steuerung von Bewegung 132
Übung 4: Servosteuerung mit Taster 135
Zusammenfassung 141

Kapitel 5: Drehbewegungen auswerten 147
Einstellknöpfe für die Gerätesteuerung 147
Das Potentiometer – ein veränderlicher Widerstand 147
Übung 1: Aufbau und Test einer Potentiometerschaltung 149
Übung 2: Widerstandsbestimmung durch Zeitmessung 151
Übung 3: Potentiometerwerte mit BASIC Stamp auslesen 158
Übung 4: Servosteuerung mit Potentiometer 162
Zusammenfassung 169

Kapitel 6: Displays ansteuern 175
Displays informieren 175
Aufbau einer 7-Segment Anzeige 175
Übung 1: Aufbau und Testschaltung mit einer 7-Segment Anzeige 177
Übung 2: Ansteuerung einer 7-Segment Anzeige 181
Übung 3: Ziffern auf Display ausgeben 184
Übung 4: Eine Drehbewegung auf Display anzeigen 191
Zusammenfassung 197

Kapitel 7: Lichtstärken messen 201
Lichtsensoren in Gebrauchsgegenständen 201
Der Fotowiderstand 201
Übung 1: Aufbau und Test eines Lichtmessgerätes 202
Übung 2: Lichtmessungen grafisch auswerten 205
Übung 3: Lichtereignisse aufzeichnen 210
Übung 4: Einfaches Lichtmessgerät 216
Zusammenfassung 229

Kapitel 8: Frequenzen und Töne 235
Ein Leben mit elektronischen Signaltönen 235
Mikrocontroller, Lautsprecher, Signaltöne und HIGH/LOW Signale 235
Übung 1: Aufbau und Test einer Schaltung mit Schallwandler 236
Übung 2: Spielgeräusche erzeugen 238
Übung 3: Noten und einfache Songs programmieren 244
Übung 4: Musik mit dem Mikrocontroller 250
Übung 5: Handy Klingeltöne programmieren 263

Zusammenfassung 276

Kapitel 9: Integrierte Schaltungen 281
Gehäuseformen 281
Projekte mit integrierten Schaltkreisen erweitern 283
Übung 1: Steuerung des Stromflusses mit einem Transistor 283
Übung 2: Schaltung mit Digitalpotentiometer 286
Zusammenfassung 296

Kapitel 10: Komplexe Systeme 301
Subsysteme integrieren 301
Übung 1: Aufbau und Test von zwei Tasterschaltungen 302
Übung 2: Aufbau und Test von zwei RC-Schaltungen 305
Übung 3: Ein Beispiel für die Integration von Subsystemen 307
Übung 4: Entwicklung eines zusätzlichen Softwarepaketes 312
Zusammenfassung 318

Anhang A: USB-Seriell Adapter 323

Anhang B: Ausstattung und Bauteile 325

Anhang C: Komponenten des BASIC Stamp Moduls und ihre Funktionen 329

Anhang D: Batterien und Netzteile 333

Anhang E: Fehlersuche 337

Anhang F: Grundlagen der Elektrizitätslehre 341

Anhang G: RTTTL Formatanweisungen 349

Index 351

Vorwort

In diesem Buch wird die Frage, “Wie arbeitet ein Mikrocontroller?” dadurch beantwortet, dass man Schülerinnen und Schülern der Mittelstufe Möglichkeiten aufzeigt, wie sie mit Hilfe des BASIC Stamp® Mikrocontroller Moduls von Parallax, Inc. eigene Ideen und Vorstellungen technologisch umsetzen können.. Die Übungen dieses Buches sind gespickt mit interessanten Experimenten, die sich mit Bewegung, Licht, Ton und Sensorik befassen und die Phantasie der Schülerinnen und Schüler anregen. Es werden Grundprinzipien aus den Bereichen Programmierung, Elektrizitätslehre, Elektronik sowie Mathematik und Physik vermittelt. In vielen Übungen werden mit kostengünstigen Bauteilen in einfach aufzubauenden Versuchen die technischen Raffinessen alltäglich genutzter Geräte und Anwendungen aufgedeckt.

HINWEIS AN DEN LESER

Die Struktur dieses Lehrbuches ist so aufgebaut, dass es sich mit seinen Inhalten sowohl an Schülerinnen und Schüler mit geringen Vorkenntnissen als auch an Hobbyisten wendet. Alle Übungen in diesem Buch sind aufeinander abgestimmt und können von den Schülerinnen und Schülern unter Anleitung einer Lehrkraft selbstständig aufgebaut, durchgeführt und ausgewertet werden; zu jedem Versuchsaufbau gibt es Hinweise, einen Schalt- und Verdrahtungsplan sowie eine umfassende Erklärung. Mit Hilfe von Verständnisfragen, Übungen und Projekten am Ende eines jeden Kapitels (jeweils mit Lösungen) lassen sich Fähigkeiten, Problemlösungsverhalten und Faktenwissen der Jungforscher abtesten. Dieses Buch ist auch für Selbstlerner, die den Lernfortschritt selbst bestimmen möchten, geeignet. Sie erhalten Hilfe über das Stamps in Class® Yahoo Group Forum, auf das weiter unten hingewiesen wird.

UNTERSTÜTZUNG UND DISKUSSIONSFOREN

Wenn Sie Unterstützung bei der Arbeit mit diesem Buch benötigen, stehen Ihnen zwei englischsprachige Yahoo! Diskussionsforen zur Verfügung. Gehen Sie dazu auf die Homepage von Parallax unter www.parallax.com und wählen Sie über das Menü *Support* unter folgenden Diskussionsforen:

Stamps In Class Group: Dieses nicht moderierte Forum wendet sich an Studenten, Lehrkräfte und Hobbyisten gleichermaßen; es werden Fragen, die sich zu den Aktivitäten, Übungen und Projekten in diesem Buch ergeben, von den Lesern dieses Forums beantwortet.

Parallax Educator's Group: Ist ein moderiertes Forum, das sich ausschließlich an Ausbilder wendet und diese in ihrer Arbeit unterstützt; Rückmeldungen zu den Stamps in Class Curricula sind erwünscht. Mitglied in dieser Benutzergruppe können nur von Parallax anerkannte Lehrkräfte und Ausbilder werden. Hinweise und Hilfen für Lehrkräfte können dazu über das Forum heruntergeladen werden.

Educational Support: stampsinclass@parallax.com Wenden Sie sich bitte an das Parallax Stamps in Class Team, wenn es Probleme beim Einschreiben in eine der Yahoo!Benutzergruppen geben sollte oder wenn es Fragen zu dem verwendeten Material in diesem Buch, das Stamps in Class Curriculum, unsere Kurse für Lehrkräfte oder allgemeine Fragen zum Lehrangebot gibt.

Schulversionen: sales@parallax.com Wenden Sie sich an unser Verkaufsteam, wenn Sie Informationen über Sonderpreise für Ausbildungsstätten und Klassenraum-Sets für unsere Stamps in Class Pakete und weitere ausgesuchte Produkte erhalten möchten.

Technische Unterstützung: support@parallax.com Bei allgemeinen Fragen zum Aufbau und Gebrauch unserer Hard- und Software wenden Sie sich bitte an unser technisches Support Team.

LEHRERHANDBUCH

Zu jedem Kapitel gibt es im Abschnitt Zusammenfassung eine Reihe von Verständnisfragen, Aufgaben und Projekte, zu denen auch Lösungen angegeben sind. Zum vorliegenden Buch wird auch ein Lehrerhandbuch angeboten, in dem zusätzliche Fragen, Aufgaben und Projekte, teilweise auch mit mehreren Lösungansätzen, vorgestellt werden. Das Lehrerhandbuch kann von Lehrkräften (Nachweis erforderlich) kostenlos über die Parallax Educators Yahoo Group oder durch Anforderung über stampsinclass@parallax.com als Worddokument oder PDF-Datei bezogen werden.

DER STAMPS IN CLASS LEHRGANG

Wie arbeitet ein Mikrocontroller? ist der Grundlagenband zu unserem Stamps in Class Lehrgang auf dem alle weiteren Bücher aufbauen. Sie sind im Internet kostenlos herunterzuladen unter der Adresse www.parallax. Zum Zeitpunkt der Drucklegung dieses Textes sind die unten angegebenen Bücher erhältlich. Überprüfen Sie bitte über unsere Website www.parallax.com oder www.stampsinclass.com, ob Sie im Besitz der neuesten

Ausgabe sind. Wir sind stets bemüht, unser pädagogisches Programm zu aktualisieren und zu verbessern.

Stamps in Class Student Guides:

Für eine solide Einführung in Entwurfstechniken und Entwurfsmethoden moderner Maschinen und Geräte ist eine gründliche Erarbeitung der Übungen und Projekte in den folgenden Anwenderhandbüchern dringend empfohlen.

"*Applied Sensors*", Student Guide, Version 1.3, Parallax Inc., 2003
"*Basic Analog and Digital*", Student Guide, Version 1.3, Parallax Inc., 2004
"*Industrial Control*", Student Guide, Version 1.1, Parallax Inc., 1999
"*Robotics with the Boe-Bot*", Student Guide, Version 2.0, Parallax Inc., 2003
"*Robotics mit dem Boe-Bot*", Handbuch, Version 2.1, Parallax Inc., 2003

Weitere Robotics Bausätze:

Erfolgt der Einstieg in den Stamps in Class Lehrgang über das Buch *Robotics mit dem Boe-Bot*, ist man für jeden der beiden folgenden Aufbautexte und Bausätze gut gerüstet:

"*Advanced Robotics: with the Toddler*", Student Guide, Version 1.2, Parallax Inc., 2003
"*SumoBot*", Manual, Version 2.0, Parallax Inc., 2004

Projektorientierte Bausätze:

Elements of Digital Logic, Understanding Signals und *Experiments with Renewable Energy* konzentrieren sich stärker auf Themen der Elektronik, während sich *StampWorks* mit seinen verschiedenen Projekten aus unterschiedlichen Anwendungsgebieten an vielseitig interessierte Entdecker, Erfinder und Entwickler wendet.

"*Elements of Digital Logic*", Student Guide, Version 1.0, Parallax Inc., 2003
"*Experiments with Renewable Energy*", Student Guide, Version 1.0, Parallax Inc., 2004
"*StampWorks*", Manual, Version 1.2, Parallax Inc., 2001
"*Understanding Signals*", Student Guide, Version 1.0, Parallax Inc., 2003

Referenz

Dieser Band ist das Referenzhandbuch für alle Stamps in Class Anwenderhandbücher. In ihm finden sich alle Informationen über die Mikrocontroller der BASIC Stamp Familie, den BASIC Stamp Editor und die Programmiersprache PBASIC.

"BASIC Stamp Manual", Version 2.0c, Parallax Inc., 2000

ÜBERSETZUNGEN

Die Parallax Lehrbücher können mit unserer Erlaubnis auch in andere Sprachen übersetzt werden (e-mail stampsinclass@parallax.com). Wenn Sie sich mit dem Gedanken tragen, eine Übersetzung durchführen zu wollen, kontaktieren Sie uns bitte, damit wir Ihnen die entsprechend formatierten MS Word Dokumente, Bilder usw. zur Verfügung stellen können. Parallax hat für seine Übersetzer ein Diskussionsforum, die Parallax Translators Yahoo-group, eingerichtet, dem Sie sich anschließen können. Hinweise, wie Sie zu diesem Forum gelangen können, finden Sie weiter unten in diesem Text. Siehe Abschnitt: WEB SITE AND DISCUSSION LISTS.

DANKSAGUNGEN

Zum Parallax Team, das an diesem Buch mitgewirkt hat, gehören: Andy Lindsay Curriculum Entwurf und Autor, Rich Allred – Illustration, Jen Jacobs und Larissa Crittenden – Cover Design, Aristides Alvarez und Jeff Martin – Unterstützung in allgemeinen Fragen, John Barrowman – Berater für den Bereich Elektromechanik, Kris Magri – Problemlöser in technischen Fragen und kritischer Freund, Stephanie Lindsay – technische Redaktion sowie Rich Allred, Gabe Duran, Stephanie Lindsay und Kris Magri – Redaktion.

Das Anwenderhandbuch *What's a Microcontroller?* Version 2.2 von Andy Lindsay entstand auf der Basis seiner eigenen Beobachtungen und der zahlreichen Rückmeldungen von Lehrkräften während seiner Tätigkeit als landesweit tätiger Parallax-Ausbildungssleiter für Fortbildungskurse für Lehrkräfte. Andy studierte Elektrotechnik und Elektronik an der California State Universität in Sacramento; dies ist sein drittes *Stamps in Class* Anwenderhandbuch. In verschiedenen Zeitschriften ist er Mitautor zu Themen aus dem Bereich Mikrocontroller in Techniklehrplänen an Schulen. Wenn er sich nicht mit dem Schreiben von Lehrbüchern beschäftigt, arbeitet Andy bei Parallax im Bereich Product Engineering.

Parallax dankt dem StampsInClass Yahoo Group Mitglied Robert Ang für die sorgfältige Durchsicht des Entwurfs und die zahlreichen Hinweise sowie dem erfahrenen Techniker und geschätzten Kunden Sid Weaver für seine verständnisvolle Kritik. Ein Dank geht an alle Stamps in Class Autoren: Tracy Allen (*Applied Sensors*) und Martin Hebel (*Industrial Control*) für ihre Anmerkungen und Ratschläge. Andy Lindsay dankt seinem Vater Marshall und seinem Schwager Kubilay für ihre fachkundige Unterstützung in allen Fragen zur Musik. *Stamps in Class* wurde von Ken Gracey ins Leben gerufen und Ken dankt allen Mitarbeitern von Parallax für ihre hervorragende Arbeit. Jeder Parallaxian hat in der einen oder anderen Form zum Erfolg dieses und aller anderen Stamps in Class Bücher beigetragen.

Kapitel 1: Erste Schritte mit der BASIC Stamp

MIKROCONTROLLER, UNSERE HEIMLICHEN HELFER

Ein Mikrocontroller ist eine Art Minicomputer, der je nach Ausführung unterschiedliche Leistungsstärken aufweist. Einige Beispiele von Alltagsgeräten aus unserer Erfahrungswelt, die mit Mikrocontrollern ausgerüstet sind, werden in Abbildung 1-1 gezeigt. Verfügen die Geräte zusätzlich über Tasten und eine Anzeige, dann kann es sich auch um programmierbare Controller handeln.

Abbildung 1-1
Beispiele von Alltagsgeräten, die mit einem Mikrocontroller arbeiten

Halte in einer Liste fest, wieviele Geräte mit einem Mikrocontroller du täglich benutzt. Hier einige Beispiele: wenn sich der Wecker des Uhrenradios meldet und du den Snooze-Schalter morgens wiederholt drückst, ist ein Mikrocontroller beteiligt; beim Simsen mit einem Handy oder dem Aufwärmen einer Speise in der Mikrowelle ist ebenfalls ein Mikrocontroller beteiligt. Damit nicht genug: wenn du mit der Fernbedienung die Fernsehkanäle durchzapst, einen Taschenrechner oder Gameboy benutzt oder die Zeit von deiner digitalen Armbanduhr abliest, sind immer Mikrocontroller irgendwo in den technischen Geräten als kleine Helfer tätig.

DIE BASIC STAMP 2

Das BASIC Stamp® 2 Modul von Parallax, Inc. (siehe Abbildung 1-2) verfügt über einen eingebauten Mikrocontroller mit Prozessorsteuerung und Ein-/Ausgabekomponenten, das auch als „eingebettetes Computer System“ oder kürzer als „Eingebettetes System“ angesprochen wird. Umgangssprachlich bezeichnet man die Module auch kurz als „Mikrocontroller“. Der Controller selbst ist der schwarze Baustein mit der Beschriftung „PIC16C57“. Die zusätzlichen Bauteile auf dem Modul sind elektronische Komponenten, die sich auch in vielen Geräten des täglichen Lebens wiederfinden.

In den Übungen dieses Buches werden wir mit Schaltungen arbeiten, die sich vielfach auch in Haushalts- und High-Techgeräten befinden. Wir werden Computerprogramme

schreiben und das BASIC Stamp Modul damit zum Leben erwecken. Mit den Programmen überwachen und kontrollieren wir unsere Schaltungen.

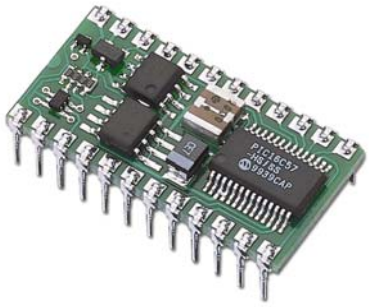


Abbildung 1-2
Das BASIC Stamp® 2 Mikrocontroller Modul

BASIC Stamp 2 Module sind die bekanntesten Mikrocontroller von Parallax, Inc.

Im vorliegenden Text bezieht sich "BASIC Stamp" auf das BASIC Stamp® 2 Mikrocontroller Modul von Parallax Inc. Es gibt weitere BASIC Stamp Module, von denen einige in Abbildung 1-3 gezeigt werden. Jedes BASIC Stamp Modul ist farbcodiert. Die BASIC Stamp 2 ist grün, die BASIC Stamp 2e rot, die BASIC Stamp 2SX blau und die BASIC Stamp 2p goldfarbig. Die unterschiedlichen Ausführungen der BASIC Stamp 2 unterscheiden sich in der Geschwindigkeit, dem Speicher und Zusatzfunktionen oder Kombinationen aus den genannten Eigenschaften.

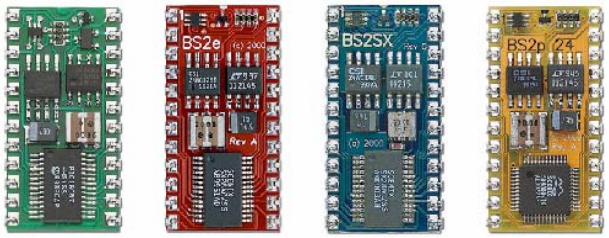


Abbildung 1-3
BASIC Stamp® Module

Von links nach rechts: BASIC Stamp® 2, 2e, 2SX, and 2p

HERAUSRAGENDE PROJEKTE MIT BASIC STAMP MIKROCONTROLLERN

Mikrocontroller werden nicht nur in Haushaltsgeräte oder Geräte des täglichen Bedarfs eingebaut. Roboter, Maschinen, Flugzeuge und viele weitere Hightech Geräte enthalten Mikrocontroller. Schauen wir uns dazu einmal einige Beispiele an, die mit BASIC Stamp Modulen entwickelt wurden.
Mit einfach aufgebauten Robotern, die unterschiedlich komplexe mechanische Funktionen ausführen können, lässt sich die Arbeitsweise von Mikrocontrollern am einfachsten verstehen. Abbildung 1-4 zeigt zwei Beispiele von Robotern, die mit Hilfe

der BASIC Stamp 2 Sensoren abfragen, Motoren steuern und mit anderen Computern Verbindung aufnehmen. Die linke Abbildung zeigt den Boe-Bot™ von Parallax Inc. Die Projekte im Übungsbuch *Robotics mit dem Boe-Bot* sollten erst nach dem Durcharbeiten der Übungen in diesem Buch angegangen werden. In der Abbildung rechts wird der Roboter einer Studentengruppe gezeigt, die an einem Roboter-Wettbewerb teilnahm. Die Ziele eines solchen Wettbewerbes ändern sich jedes Jahr; in dem vorliegenden Beispiel mussten die Roboter möglichst schnell Ringe farblich sortieren.

Abbildung 1-4
Ausbildungsroboter

Parallax Boe-Bot™ (links)
First Competition Robot (rechts)

Andere Roboter wiederum lösen komplexe Probleme, wie bei dem ferngesteuerten Flugroboter in Abbildung 1-5, links. Dieser Roboter wurde von Ingenieurstudenten der Universität von Californien, Irvine aufgebaut und getestet. Sie benutzten dazu ein BASIC Stamp Modul als Hilfe für die Kommunikation mit dem GPS-System. Damit konnte der Roboter jederzeit seine Position und Höhe bestimmen. Die BASIC Stamp las auch Kontrollsensoren aus und steuerte die Motoreneinstellungen um die Flugeigenschaften des Roboters zu optimieren. Der mechanische Tausendfüßler-Roboter im rechten Bild wurde von einem Professor der technischen Nanyang Universität in Singapur entwickelt. Er verfügt über 50 BASIC Stamp Module, die alle über ein ausgetüfteltes Netzwerk miteinander verbunden sind und dadurch die Kontrolle und das Zusammenspiel der Bewegung jedes Beines erst ermöglichen. Mit Hilfe solcher Roboter lassen sich Bewegungsabläufe in der Natur besser verstehen, sie tragen aber auch dazu bei, die Phantasie anzuregen.Vielleicht ist es mit Hilfe von Robotern zukünftig möglich schwer zugängliche Orte oder andere Planeten zu erkunden.

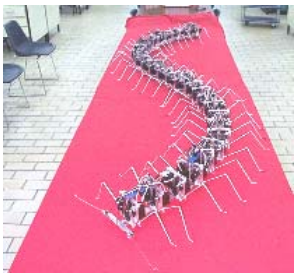


Abbildung 1-5
Beispiele von Robotern

Selbstständig fliegendes Objekt der Uni Californien Irvine (links)und dasTausend-füßlerprojekt der Uni Nanyang (rechts)

Mit Hilfe von Mikrocontrollern werden Roboter zukünftig alltägliche Arbeiten übernehmen, wie z.B. den Rasen mähen. Das BASIC Stamp Modul des Rasenmäher-Roboters aus Abbildung 1-6 überwacht, dass der Mäher die Rasenfläche nicht verlässt. Über Sensoren werden Hindernisse erkannt und die Motoren des Mähers gesteuert.

Abbildung 1-6
Robotermodell
Rasenmäher

Prototyp des Robot Shop

Mikrocontroller werden aber auch in wissenschaftlichen, hochtechnologischen und Luftfahrtprojekten eingesetzt. Die Wetterstation aus Abbildung 1-7(links) sammelt zum Beispiel Umweltdaten, aus denen sich Aussagen über die Lebensdauer eines Korallenriffs ableiten lassen. Das eingebaute BASIC Stamp Modul sammelt über verschiedene Sensoren Daten und speichert sie zur späteren Auswertung ab. Das U-Boot in der Bildmitte wird mit seinen controllergesteuerten Servomotoren, Kameras und Lichtquellen für Unterwasseruntersuchungen eingesetzt. Die Rakete rechts im Bild nahm an einem Wettbewerb teil, bei dem es darum ging, eine von Amateuren gebaute Rakete in den Weltraum zu schießen. Es gab keinen Wettbewerbsgewinner, obwohl diese Rakete nah am Ziel war. Eine BASIC Stamp übernahm die Überwachung aller Vorgänge beim Start.

Abbildung 1-7
Beispiele für den Einsatz von Mikrocontrollern in High-tech und Luftfahrt-projekten

Ökologische Datenauf-nahme durch EME Systems (links), Unterwasser-forschung des Harbor Branch Institut (Mitte), und JP Aerospace Teststart (rechts)

Von einfachen Anwendungen aus dem Haushalt über wissenschaftliche und luftfahrttechnische Anwendungen werden alle Grundlagen des Mikrocontrollereinsatzes, die zum Verständnis notwendig sind, in diesem Buch vermittelt. Die Übungsteile dieses Buches sollen zum Experimentieren anregen und gleichzeitig das Wissen vermitteln, wie Schaltungen, die sich heute in jedem High-tech Gerät finden lassen, arbeiten. Wir werden Schaltkreise aufbauen für Displays, Sensoren und zur Steuerung von Bewegung. Wir werden lernen, wie man diese Schaltkreise mit einem BASIC Stamp 2 Modul verbindet und wir werden Computerprogramme schreiben, mit dem Displays angesteuert, Daten aus Sensoren ausgelesen und Bewegungen gesteuert werden. So ganz nebenbei wirst du viel über elektronische Schaltungs- und Programmiertechnik lernen und es wird die Zeit kommen, dass du eigene Projekte entwickelst.

HARDWARE UND SOFTWARE

Der Einstieg in die Welt der BASIC Stamp Mikrocontroller gestaltet sich in etwa so, wie die Inbetriebnahme eines neuen PC oder Laptops. Zuerst wird er ausgepackt, an die Steckdose angeschlossen, etwas Software installiert und dann wird ausprobiert und, wenn man zusätzlich über Programmierkenntnisse verfügt, vielleicht ein erstes kleines Programm in einer Programmiersprache entworfen. Wer das erste Mal mit einem BASIC Stamp Modul arbeitet, wird genau diese Dinge tun. Arbeitet man im Klassenverband, sind die Installations- und Einstellungsarbeiten bereits abgeschlossen und der Lehrer gibt Hinweise was zu tun ist. Falls dies nicht der Fall ist, wird im folgenden Kapitel erklärt, welche Schritte auszuführen sind, damit die neue BASIC Stamp das Laufen lernt.

ÜBUNG 1: DER BASIC STAMP EDITOR

In den meisten Übungen und Projekten dieses Buches wird der BASIC Stamp Editor (Version 2.0 oder höher) eingesetzt. Mit dieser Software werden Programme geschrieben, die das BASIC Stamp Modul abarbeitet. Natürlich lassen sich damit auch Mitteilungen auf einer Anzeige ausgeben; das kann mit dazu beitragen, die Arbeitsweise der BASIC Stamp besser zu verstehen.

Der BASIC Stamp Editor ist Gratissoftware. Es gibt zwei Möglichkeiten, ihn zu erwerben:

- **Download aus dem Internet:** www.parallax.com → Downloads → BASIC Stamp Software → "BASIC Stamp Windows Editor version 2.0…" .
- **Parallax CD:** Folge dem Link auf der Welcome Seite. Überprüfe, ob das Datum auf der CD May 2003 oder später ist.

In Eile? Besorge dir eine Kopie des BASIC Stamp Windows Editors Version 2.0 (oder höher) und installiere sie auf dem PC oder Laptop. Gehe dann direkt zur Übung 3: hardware einrichten und systemtest.

Wenn noch Fragen sind, kann Übung 1 bei der Beschaffung der Software als Schritt für Schritt Anleitung und Übung 2 als Referenz für die Installation der Software genutzt werden.

Systemvoraussetzungen

Der BASIC Stamp Editor läuft sowohl auf einem PC als auch auf einem Laptop mit den folgenden Systemvoraussetzungen:

- Betriebssystem Microsoft Windows 95® oder höher
- Eine serielle Schnittstelle oder ein USB Port
- Ein CD-ROM Laufwerk und/oder Internetzugang

USB Adapter Verfügt der Rechner nur über USB Anschlüsse, wird ein USB-Seriell Adapter benötigt. Einzelheiten und Installationshinweise findest du im Anhang A: USB-Seriell Adapter .

Software aus dem Internet herunterladen

Das Herunterladen der BASIC Stamp Editor Software von der Parallax Website gestaltet sich einfach. Abbildung 1-8 zeigt die Parallax Website, die inzwischen auch anders aussehen kann. Zum Herunterladen der Software gehst du wie folgt vor:

√ Rufe mit Deinem Browser über www.parallax.de die Website von Parallax auf (siehe Abbildung 1-8).
√ Öffne über das Menü *Downloads* das Optionenmenü.
√ Wähle *BASIC Stamp Software*.

Abbildung 1-8
Die Parallax Web Site:

www.parallax.com

√ Wähle auf der „BASIC Stamp Software“-Seite die neueste Version des BASIC Stamp Editors aus; die Versionsnummer sollte 2.0 oder höher sein.
√ Weiter mit *Download*. In Abbildung 1-9 ist das Downloadsymbol in der Spalte Download als Aktenordner dargestellt und befindet sich rechts neben dem Text: “BASIC Stamp Windows Editor version 2.0 Beta 1 (6MB)”.

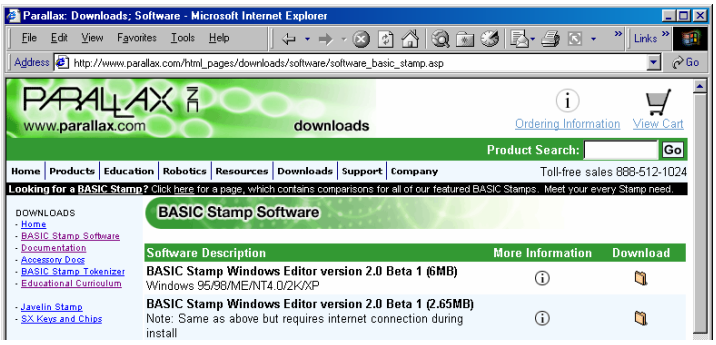


Abbildung 1-9
Die Parallax Web Site Download Seite

√ Sobald das Fenster *File Download* erscheint (siehe Abbildung 1-10), wähle: *Save this program to disk.*
√ Weiter mit *OK*.

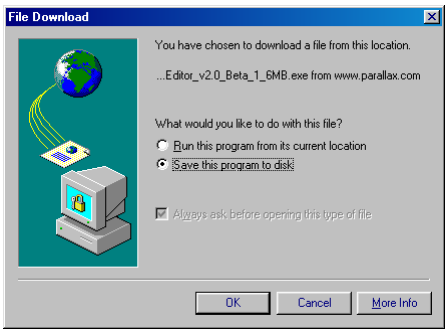

File Download
You have chosen to download a file from this location.
...Editor_v2.0_Beta_1_6MB.exe from www.parallax.com
What would you like to do with this file?
Run this program from its current location
Save this program to disk
Always ask before opening this type of file
OK
Cancel
More Info

Abbildung 1-10
File Download Fenster

Als nächstes erscheint das in Abbildung 1-11 dargestellte Fenster *Speichern unter*. Über das Feld *Speichern in* lässt sich die Datei in einem Verzeichnis der eigenen Wahl abspeichern.

√ Nachdem der Speicherort bestimmt wurde, wähle *Speichern*.

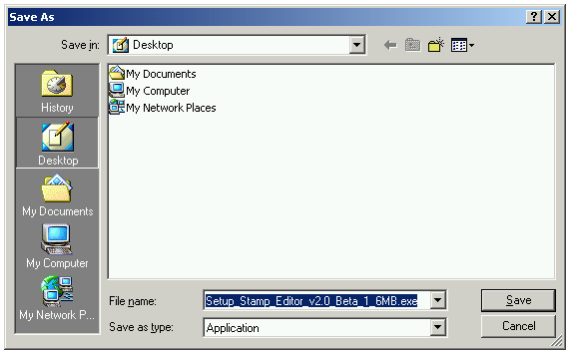


Abbildung 1-11
Save As Fenster

Lege den Speicherort für die Datei fest.

√ Warte so lange, bis der komplette BASIC Stamp Editor auf den Rechner übertragen wurde (siehe Abbildung 1-12). Der Vorgang kann je nach Internetzugang etwas länger dauern.

√ Schließe nach dem Herunterladen der Software das Fenster (siehe Abbildung 1-13) noch nicht sondern gehe direkt zum nächsten Abschnitt - Übung 2: Installation der Software.

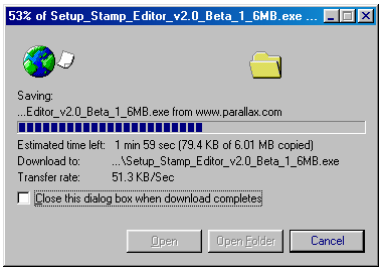


Abbildung 1-12: Fortschrittsfenster beim Herunterladen

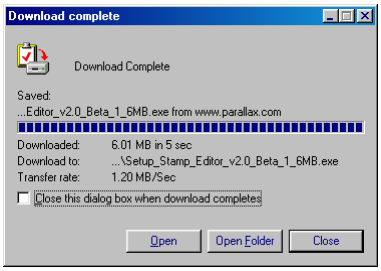


Abbildung 1-13: Herunterladen abgeschlossen

Die Software auf der Parallax CD

Die BASIC Stamp Editor Software kann auch direkt von der Parallax CD installiert werden. Achte bitte darauf, dass die Version nicht älter als vom Mai 2003 ist, damit wird sicher gestellt, dass der Editor kompatibel zu den Programmbeispielen in diesem Buch ist. Die Monats- und Jahresangabe ist auf der Parallax CD aufgedruckt.

- √ Lege die Parallax CD in das CD-Laufwerk des Rechners. Nach kurzer Zeit erscheint das Eröffnungsbild Welcome application (siehe Abbildung 1-14).
- √ Sollte das Eröffnungsbild nicht automatisch erscheinen, wähle nacheinander *Arbeitsplatz*, *Disk (D:* <oder anderes Laufwerk>) und die Datei *Welcome*.

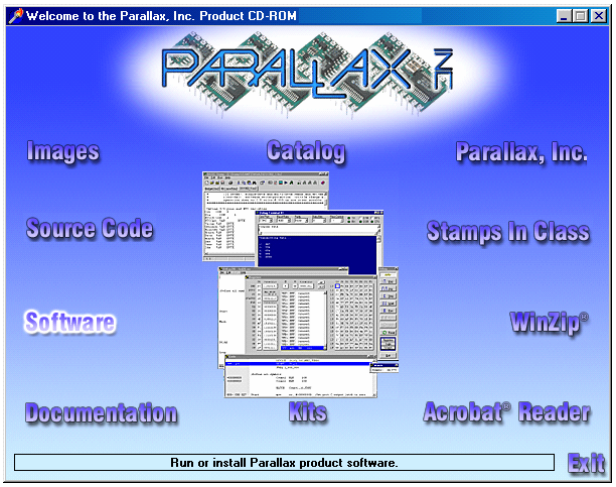


Abbildung 1-14
Der Parallax CD Browser

√ Weiter mit *Software,* siehe Abbildung 1-14.
√ Aktiviere + neben dem Ordner *BASIC Stamps,* siehe Abbildung 1-15.
√ Aktiviere + neben dem Ordner *Windows*.
√ Aktiviere das Floppy Disk Symbol "Stamp 2/2e/2sx/2p/2pe (stampw.exe)".
√ Mache weiter mit Übung 2: Installation der Software.

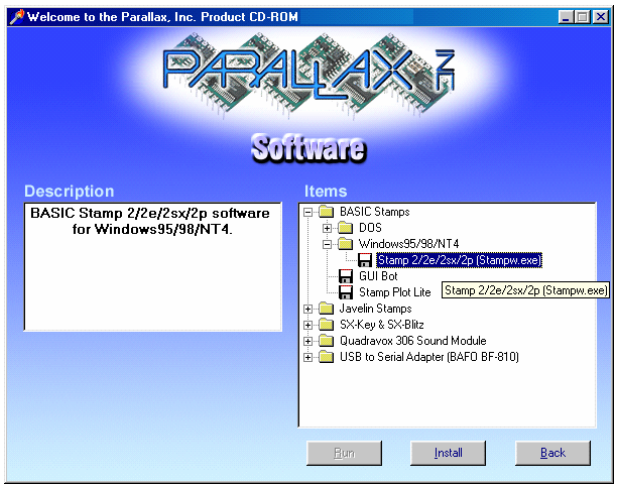


Abbildung 1-15
Der Parallax CD Browser

Wähle das BASIC Stamp Editor Installationsprogramm von der Software Seite.

Die kostenlosen Downloadangebote auf der Parallax Website befinden sich auch auf der Parallax CD. Auf der CD ist zur Orientierung das Erstellungsdatum aufgedruckt. Die Dateien auf der Website sind häufig aktueller als die Dateien, die auf CD gebrannt wurden. Sollte die CD bereits mehrere Monate alt sein, kann über die Website schnell überprüft werden, ob man über die letzte Programmversion verfügt und sie gegebenenfalls von der Parallax Website neu herunterladen oder eine neue CD bestellen.

ÜBUNG 2: INSTALLATION DER SOFTWARE

Das Installationsprogramm für den BASIC Stamp Editor wurde inzwischen entweder aus dem Internet von der Parallax Website heruntergeladen oder auf der Parallax CD gefunden. Im nächsten Schritt wird das Installationsprogramm gestartet.

Installation der Software Schritt für Schritt

√ Ist das Installationsprogramm des BASIC Stamp Editors aus dem Internet heruntergeladen, geht es weiter im Fenster *Download Complete*; aktiviere *Open* (siehe Abbildung 1-16).

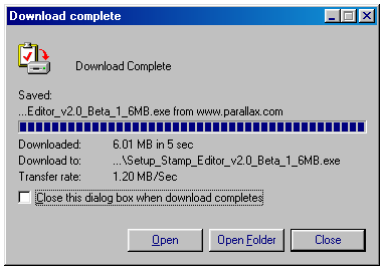


Abbildung 1-16
Download Complete Fenster

Klicke auf Open.

√ Wird die Software von der Parallax CD installiert, klicke auf *Install* wie in Abbildung 1-17 dargestellt.

Abbildung 1-17
Der Parallax CD Browser

Aktiviere Install am unteren Rand des Fensters.

√ Weiter geht es im geöffneten Fenster des *BASIC Stamp Editor InstallShield Wizards* mit *Next* (siehe Abbildung 1-18).

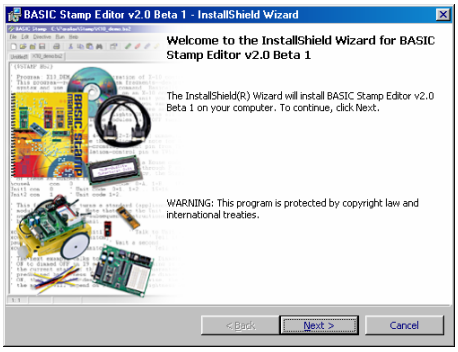

BASIC Stamp Editor v2.0 Beta 1 - InstallShield Wizard
Welcome to the InstallShield Wizard for BASIC Stamp Editor v2.0 Beta 1
The InstallShield(R) Wizard will install BASIC Stamp Editor v2.0 Beta 1 on your computer. To continue, click Next.
WARNING: This program is protected by copyright law and international treaties.
< Back
Next >
Cancel

Abbildung 1-18
InstallShield Wizard des BASIC Stamp Editors

Weiter mit Next.

√ Wähle als Setup-Einstellung *Typical* (siehe Abbildung 1-19).
√ Aktiviere *Next*.

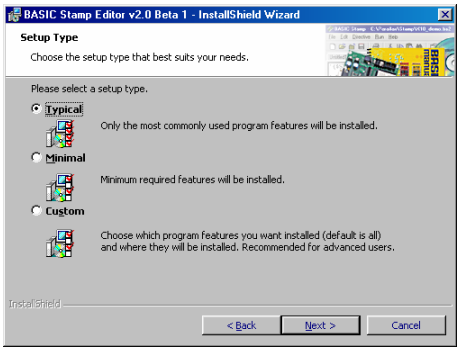

BASIC Stamp Editor v2.0 Beta 1 - InstallShield Wizard
Setup Type
Choose the setup type that best suits your needs.
Please select a setup type.
Typical
Only the most commonly used program features will be installed.
Minimal
Minimum required features will be installed.
Custom
Choose which program features you want installed (default is all) and where they will be installed. Recommended for advanced users.
InstallShield
< Back
Next >
Cancel

Abbildung 1-19
Setup Type

Wähle Typical, dann weiter mit Next.

√ Sobald der *InstallShield Wizard* die Meldung “Ready to Install the Program” anzeigt, geht es weiter mit *Install* (siehe Abbildung 1-20).

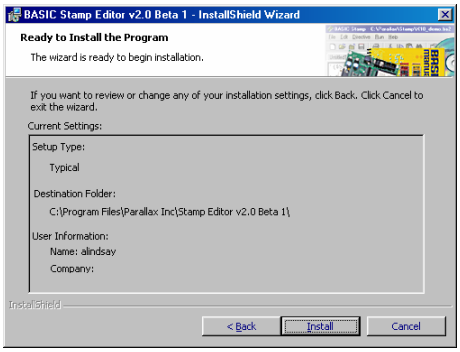

BASIC Stamp Editor v2.0 Beta 1 - InstallShield Wizard
Ready to Install the Program
The wizard is ready to begin installation.
If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.
Current Settings:
Setup Type:
Typical
Destination Folder:
C:\Program Files\Parallax Inc\Stamp Editor v2.0 Beta 1\
User Information:
Name: alindsay
Company:
InstallShield
< Back
Install
Cancel

Abbildung 1-20
Ready to Install.

Klicke auf Install.

√ Sobald im Fenster des *InstallShield Wizard* die Meldung "InstallShield Wizard Completed" angezeigt wird (siehe Abbildung 1-21), klicke auf *Finish*.

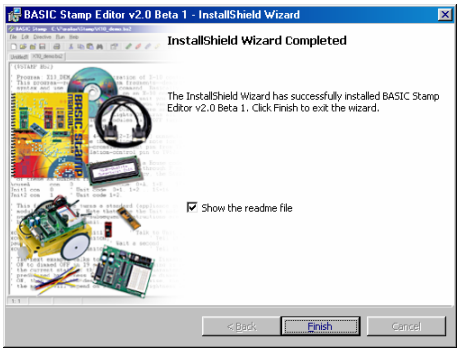

BASIC Stamp Editor v2.0 Beta 1 - InstallShield Wizard
InstallShield Wizard Completed
The InstallShield Wizard has successfully installed BASIC Stamp Editor v2.0 Beta 1. Click Finish to exit the wizard.
Show the readme file
< Back
Finish
Cancel

Abbildung 1-21
InstallShield Wizard Completed:

Klicke auf Finish.

ÜBUNG 3: HARDWARE EINRICHTEN UND SYSTEMTEST

Das BASIC Stamp Modul ist nur dann betriebsbereit und kann programmiert werden, wenn es mit einer Spannungsquelle und einem PC verbunden ist. Stelle die notwendigen Verbindungen her, rufe den *BASIC Stamp Editor* auf und teste das System. Die nächste Übung zeigt wie es geht.

Die BASIC Stamp®, das Board of Education® und das HomeWork Board™

In Abbildung 1-22 wird die Trägerplatine Board of Education® (BoE) und ein BASIC Stamp Modul gezeigt. Wie bereits weiter oben erwähnt, ist eine BASIC Stamp ein sehr kleiner Computer, der auf der Trägerplatine BoE untergebracht ist. Wir werden gleich sehen, dass sich ein BoE sehr einfach mit einer Spannungsquelle und seriellen

Schnittstelle verbinden lässt. In weiteren Übungen werden wir zeigen, wie man elektronische Schaltungen auf dem BoE aufbaut und mit dem BASIC Stamp Modul verbindet.

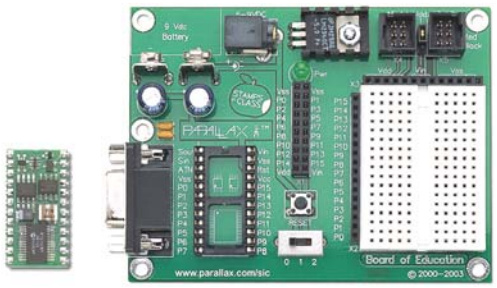


Abbildung 1-22
BASIC Stamp® 2
Mikrocontroller Modul (links)
und die Trägerplatine Board of
Education ® (rechts)

In Abbildung 1-23 ist das BASIC Stamp HomeWork Board™ als Projektplattform dargestellt. Dieses Board entspricht dem Board of Education allerdings mit einem fest integrierten BASIC Stamp 2 Modul. Seine miniaturisierten Bauelemente (SMD) sieht man auf der linken Seite des weißen Steckbretts. Für die Übungen in diesem Buch kann entweder das Board of Education mit dem BASIC Stamp Modul oder das BASIC Stamp HomeWork Board als Projektplattform benutzt werden.

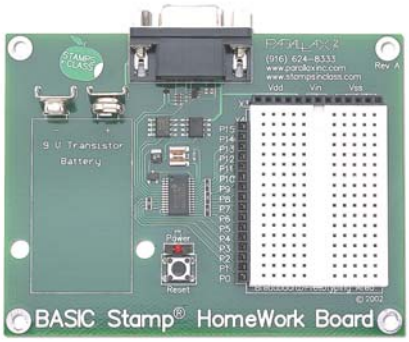


Abbildung 1-23
BASIC Stamp®
HomeWork Board™
Projektplattform.

Mehr erfährst du über die Eigenschaften und Funktionen der BASIC Stamp Module, der Trägerplatine Board of Education und der Projektplattform HomeWork Board im Anhang C: Komponenten des BASIC Stamp Moduls und ihre Funktionen auf Seite 329.

Benötigte Hardware

(1) BASIC Stamp 2 Modul UND
(1) Board of Education
- oder -
(1) BASIC Stamp HomeWork Board

(1) 9 V Batterie
(1) Streifen mit 4 Gummifüßen
(1) Serielles Kabel

Starte mit einem neuer 9 V Batterie oder einem frisch geladenen 9 V Batterieakku.

ACHTUNG! Vor dem Anschluss eines AC Adapters oder einer Gleichstrom-versorgung:

√ Im Anhang D: Batterien und auf Seite 333 kannst du überprüfen, ob die Spannungsversorgung für die vorgesehenen Übungen in diesem Buch ausreichend ist.

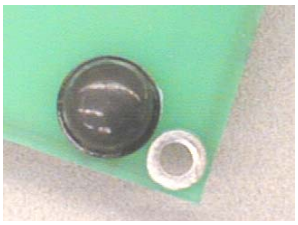
Anschluss der Hardware

Zum Board of Education und dem BASIC Stamp HomeWork Board wird ein Streifen mit vier Gummifüßen mitgeliefert (siehe Abbildung 1-24), die auf die Unterseite der Platine geklebt werden.

Abbildung 1-24
Gummifüße

√ Klebe die Gummifüße des Trägerstreifens auf die Unterseite deines Boards, so wie in Abbildung 1-25 gezeigt. Beim Board of Education sind die Klebestellen durch Kreise markiert; beim HomeWork Board klebe die Gummifüße möglichst nah an die in den vier Ecken befindlichen Bohrlöcher.

Abbildung 1-25 Gummifüße auf der Unterseite des Board of Education (links) und am HomeWork Board (rechts)



Im nächsten Schritt sollte das *Board of Education* oder das *BASIC Stamp HomeWork Board* mit dem PC oder Laptop über die serielle Schnittstelle verbunden werden.

√ Verbinde das Kabel mit einer seriellen Schnittstelle (COM1...COM4) des Computers, so wie in Abbildung 1-26 dargestellt.

USB Converter Wird ein USB-Seriell-Converter benutzt :

√ Verbinde den USB-Stecker mit der USB-Schnittstelle des Rechners.

√ Verbinde den SUB-D9 Stecker entweder direkt mit dem Board of Education oder dem HomeWork Board oder verbinde ihn mit dem seriellen Kabel, wie in Abbildung 1-26 dargestellt.

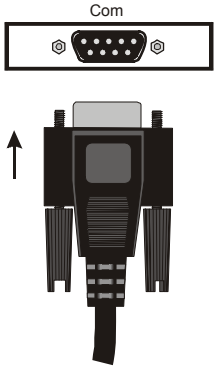


Abbildung 1-26: serielle Schnittstelle (COM-Port) am PC oder Laptop

Verbinde das serielle Kabel mit dem COM-Port deines Rechners oder Laptops.

Abb. 1-27: 3-fach Schiebeschalter

In Position 0 ist die Spannungsversorgung abgeschaltet.

Wird das *BASIC Stamp 2 Modul* mit einem *Board of Education* genutzt:

√ Bringe den 3-fach Schiebeschalter auf dem Board of Education in Position 0, so wie in Abb. 1-27 dargestellt.

Nur das Board of Education Rev C verfügt über einen 3-fach Schiebeschalter. Um bei einem Board of Education Rev B die Versorgungsspannung abzuschalten, muss die Spannungsquelle (Batterie) entfernt oder die Netzanschlussverbindung des Steckernetzteils gelöst werden. Siehe dazu Abbildung 1-28, Schritte 3 oder 4.

√ Sollte sich das BASIC Stamp Modul noch nicht auf dem Board of Education befinden, setze es jetzt in die Fassung ein. Achte dabei auf die Orientierung des Moduls wie in Abbildung 1-28, Schritt 1 dargestellt. Achte ebenso darauf, dass die Kontaktbeine des Moduls beim Einsetzen in die Fassung nicht abknicken.

√ Verbinde das serielle Kabel mit dem Board of Education, so wie in Schritt 2 dargestellt.

√ Verbinde eine Gleichspannungsquelle mit der DC Einbaubuchse wie in Schritt 3 dargestellt oder schließe einen 9 V Block an den entsprechenden Batterieclip an (siehe Schritt 4).

√ Bringe den 3-fach Schiebeschalter in Position 1. Auf dem Board of Education zeigt ein grünes Licht an, dass Spannung anliegt.

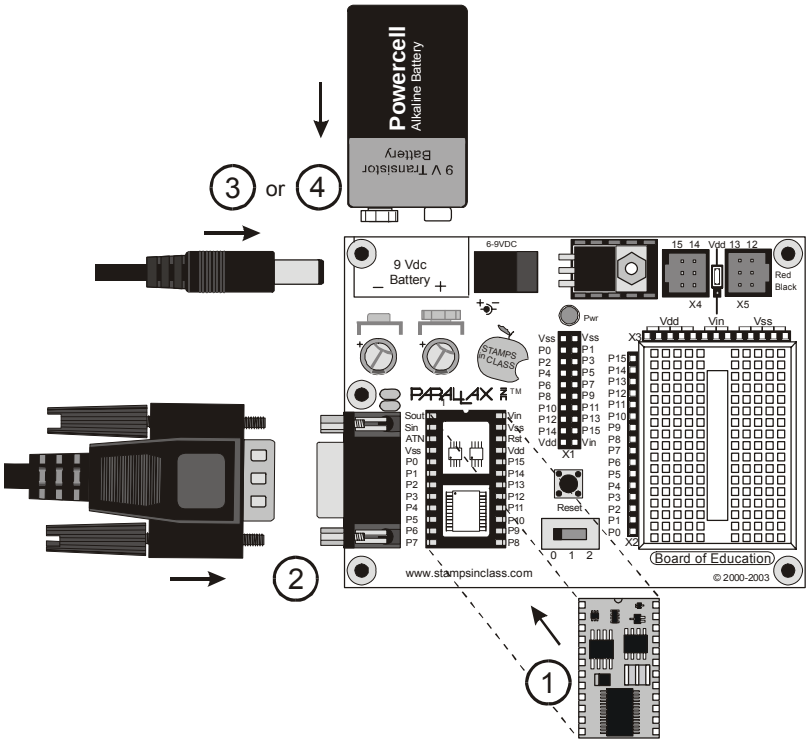


Abbildung 1-28
Board of Education, BASIC Stamp Modul, Batterie und serielles Kabel.

Verbinde die Komponenten in der dargestellten Reihenfolge. Achte darauf, dass die Einkerbung auf dem BASIC Stamp Modul zum Batteriehalter weist.

Für das BASIC Stamp HomeWork Board gilt entsprechend:

- √ Verbinde das serielle Kabel mit dem HomeWork Board (Abbildung 1-29, Schritt 1).
- √ Verbinde einen 9 V Block mit dem Batterieclip, wie in Schritt 2 dargestellt.

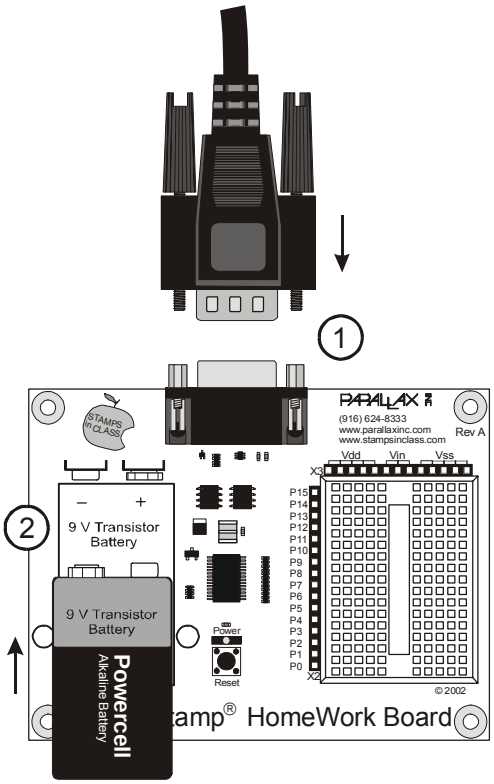


Abbildung 1-29
HomeWork Board und serielles Kabel

Verbinde das serielle Kabel und den 9 V Block mit dem jeweiligen Anschluss auf dem HomeWork Board.

Testen der Verbindung

Mit dem BASIC Stamp Editor überprüfen wir jetzt, ob der Rechner oder Laptop mit unserem BASIC Stamp Modul zusammenarbeitet.

√ Starte den BASIC Stamp Editor mit einem Doppelklick auf das Programm-Icon. Es sollte ähnlich aussehen wie in Abbildung 1-30.

Abbildung 1-30
BASIC Stamp Editor Shortcut

Auf dem Desktop des Rechners sollte dieses Icon irgendwo abgelegt sein.

Der *BASIC Stamp Editor* läßt sich auch über das **Windows Startmenü** aufrufen. Über *Start – Programme – Parallax, Inc. – BASIC Stamp Editor 2* … und anklicken des *BASIC Stamp Editor* Symbols wird das Programm gestartet.

Das *BASIC Stamp Editor*-Fenster sollte ähnlich aussehen, wie in Abbildung 1-31 gezeigt.

Beim ersten Aufruf des BASIC Stamp Editors können verschiedene Meldungen und eine Liste der gefundenen COM-Ports angezeigt werden.

√ Um sicher zu stellen, dass das BASIC Stamp Modul mit dem angeschlossenen Computer zusammenarbeitet, wähle aus dem Menü *Run* den Befehl *Identify*.

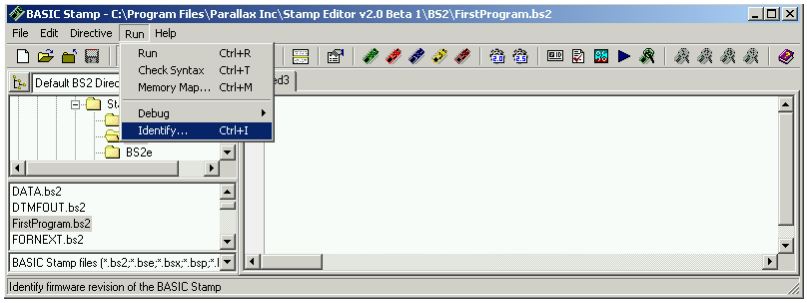


Abbildung 1-31
BASIC Stamp Editor

Wähle aus dem Menü Run den Befehl Identify.

Es erscheint ein Identifikationsfenster, ähnlich wie in Abbildung 1-32 dargestellt. Im dargestellten Beispiel hat ein BASIC Stamp 2 Modul eine COM2-Schnittstelle erkannt.

√ Überprüfe am Identifikationsfenster, ob das BASIC Stamp 2 Modul von einer COM-Schnittstelle erkannt worden ist. Ist dies der Fall, mache weiter mit Übung 4: Ein erstes Program.

√ Wird das BASIC Stamp 2 Modul von keiner COM-Schnittstelle erkannt, gibt es Hilfestellung im Anhang E: auf Seite 337.

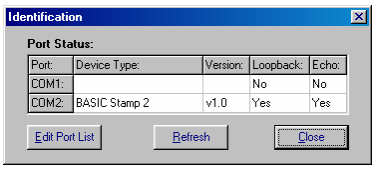


Abbildung 1-32
Identification Fenster

Beispiel: BASIC Stamp 2 an COM2.

ÜBUNG 4: EIN ERSTES PROGRAMM

Mit dem ersten selbstgeschriebenen und ausgetesteten Programm wird die BASIC Stamp angewiesen einem Rechner oder Laptop eine Nachricht zu schicken. In Abbildung 1-33 wird gezeigt, wie es gelingt, durch einen Folge aus Nullen und Einsen, die vom Mikrocontroller an den Rechner gesandt wird, Buchstaben auf dem Monitor darzustellen. Die Nullen und Einsen bezeichnen wir als binäre Zahlen. Wie wir gleich sehen werden, hat der BASIC Stamp Editor die Fähigkeit, diese Mitteilungen zu erkennen und anzuzeigen.

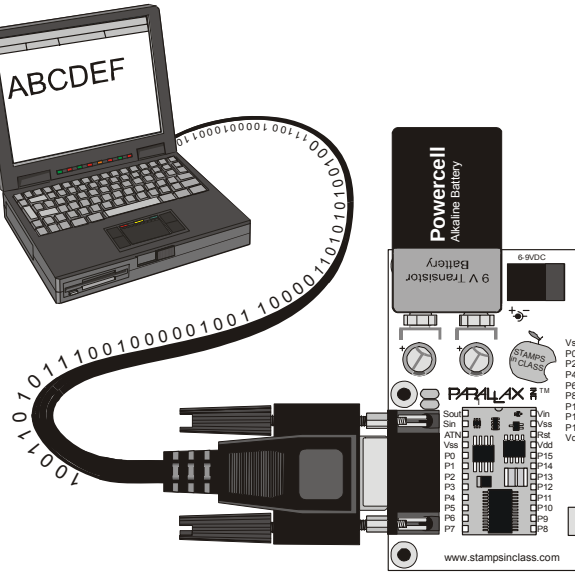


Abbildung 1-33
Mitteilungen von dem BASIC Stamp Modul an deinen Computer

Die BASIC Stamp überträgt Zeichen auf deinen PC oder Laptop, indem sie eine Folge binärer Zeichen (0 und 1) überträgt. Der BASIC Stamp Editor erkennt die binären Codes und wandelt sie in für uns lesbare Zeichen um.

Das erste Programm

Programme, die mit dem BASIC Stamp Editor geschrieben wurden und an die BASIC Stamp übertragen werden, sind immer mit einem grauen Hintergrund unterlegt, wie im folgenden Beispiel:

Programmbeispiel: FirstProgram.bs2

```
' What's a Microcontroller - FirstProgram.bs2
' BASIC Stamp sends message to Debug Terminal.

' {$STAMP BS2}
```

```
' {$PBASIC 2.5}

DEBUG "Hello, it's me, your BASIC Stamp!"
END
```

Gib dieses Programm in den BASIC Stamp Editor ein. Es werden einige Programmzeilen automatisch eingetragen, indem man die entsprechenden Schalter der Werkzeugleiste anklickt. Andere Zeilen wiederum müssen direkt über die Tastatur eingegeben werden.

- √ Aktiviere das BS2-Symbol (grüner Chip) in der Werkzeugleiste (siehe Abbildung 1-34). Wird der Cursor über diesen Schalter bewegt, erscheint die Kurzbeschreibung "Stamp Mode: BS2".
- √ Aktiviere als nächstes das Zahnrad-Symbol mit der Bezeichnung "2.5" wie in Abbildung 1-35 gezeigt. Die Kurzbeschreibung zeigt an:"PBASIC Language: 2.5".

Abbildung 1-34
BS2 Symbol

Ein Klick auf diesen Schalter fügt automatisch den Text " ' {$STAMP BS2}" *am Anfang des Programms ein.*

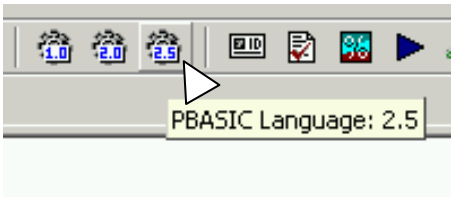


Abbildung 1-35
PBASIC 2.5 Symbol

Ein Klick auf diesen Schalter fügt automatisch den Text " ' {$PBASIC 2.5}" am Anfang des Programms ein.

Füge immer mit diesen beiden Tasten aus der Werkzeugleiste die oben genannten zwei Zeilen am Anfang eines Programmes ein!

Compilerdirektiven müssen in geschweiften Klammern { } eingegeben werden. Bei runden ( ) oder eckigen [ ] Klammern läuft das Programm nicht.

- √ Gib die noch fehlenden vier Zeilen des Programms genau so ein, wie in Abbildung 1-36 dargestellt.

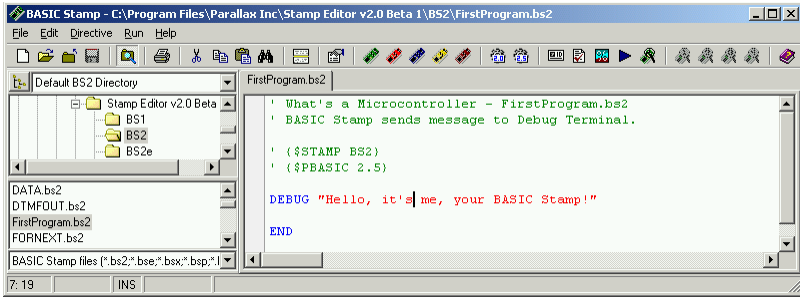


Abbildung 1-36 BASIC Stamp Editor mit dem Programm *First Program*

Gib ein erstes Programm genau so ein, wie hier dargestellt.

√ Speicher die Arbeit unter *File – Save* ab, so wie in Abbildung 1-37 gezeigt.
√ Gib im Feld *Dateiname* im unteren Teil des *Speichern unter* - Fensters *FirstProgram* ein (siehe Abbildung 1-38).
√ Weiter mit Klick auf *Speichern*.

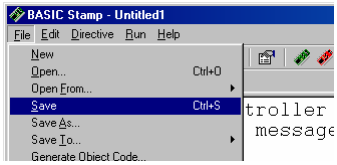


Abbildung 1-37: Speichern von *FirstProgram*

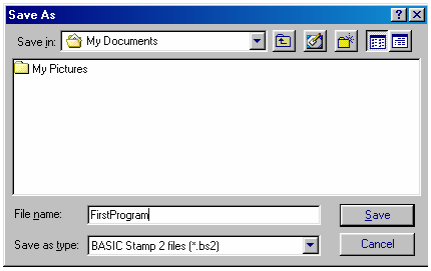


Abbildung 1-38: Eingabe des Dateinamens

Der BASIC Stamp Editor speichert Programme über *Datei – Speichern* oder das Diskettensymbol im Hauptmenü beim nächsten Mal automatisch unter dem selben Dateinamen ab (FirstProgram.bs2), solange kein anderer Name über *Datei – Speichern unter* eingegeben wird.

√ Aktiviere *Run* und wähle aus dem sich öffnenden Menü *Run* (siehe Abbildung 1-39).

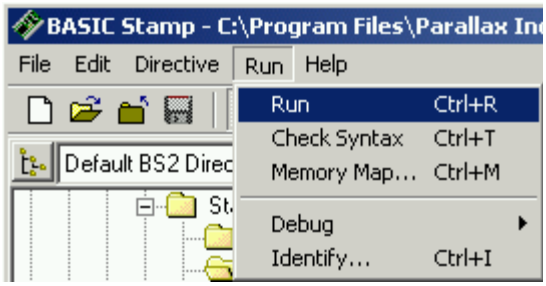


Abbildung 1-39
Start des Programms
FirstProgram

Während das Programm vom PC oder Laptop zur BASIC Stamp übertragen wird, erscheint für einen kurzen Moment ein *Download Progress* Fenster. Abbildung 1-40 zeigt das *Debug Terminal* Fenster, das sich nach Abschluss der Datenübertragung öffnet. Als Nachweis, dass die Meldung von der BASIC Stamp stammt, drücke den Reset-Schalter auf dem Board. Nach jedem Reset wird das Programm neu gestartet und es erscheint eine weitere Meldung im *Debug Terminal* Fenster.

√ Drücke auf *Reset*. Eine erscheint eine zweite „Hello …" Meldung im *Debug Terminal* Fenster.

Abbildung 1-40
Debug Terminal

Das Debug Terminal zeigt Meldungen an, die vom BASIC Stamp Modul an den PC/Laptop übertragen wurden.

Für die am häufigsten durchzuführenden Aufgaben stellt der BASIC Stamp Editor sogenannte Kurzwahltasten bereit. Um z.B. ein Programm auszuführen, genügt es die Tasten ‚Strg' und 'R' gleichzeitig zu drücken oder *Run* zu aktivieren. Es ist der Schalter mit dem kleinen blauen Dreieck wie er in Abbildung 1-41 zu sehen ist. Wird der Cursor über den Schalter *Run* bewegt, erscheint eine Kurzbeschreibung; Hinweise erscheinen auch bei jedem anderen Schalter.

Abbildung 1-41

BASIC Stamp Editor
Kurzwahltasten-Leiste

Wie funktioniert FirstProgram.bs2?

Die ersten beiden Zeilen sind sogenannte Kommentare. Ein Kommentar ist eine Textzeile, die vom BASIC Stamp Editor nicht verarbeitet wird; sie dient nur dazu, den Programmcode für den Entwickler verständlicher zu machen. Eine Kommentarzeile beginnt immer mit einem Apostroph. Die erste Kommentarzeile in *FirstProgram.bs2* informiert darüber, aus welchem Buch das Beispielprogramm stammt und unter welchem Dateinamen es abgespeichert ist. Die zweite Zeile enthält eine kurze einzeilige Beschreibung über die Wirkung des Programms.

```
' What's a Microcontroller - FirstProgram.bs2
' BASIC Stamp sends message to Debug Terminal.
```

In Kommentarzeilen werden auch sogenannte Compilerdirektiven abgelegt. Jedes Programm in diesem Buch benötigt zwei Direktiven:

```
' {$STAMP BS2}
' {$PBASIC 2.5}
```

Die erste Direktive $STAMP teilt dem BASIC Stamp Editor mit, dass das Programm an ein BASIC Stamp 2 Modul übertragen wird. Die zweite Direktive $PBASIC weist den BASIC Stamp Editor darauf hin, dass die Programmiersprache PBASIC in der Version 2.5 benutzt wird. Beachte, dass Compilerdirektiven immer in geschweiften Klammern {}eingeschlossen sein müssen. Zur Vermeidung von Eingabefehler setze die Compilerdirektiven immer über die Symbole aus der Werkzeugleiste. Werden Compilerdirektiven von Hand eingetippt, schleichen sich leicht Fehler ein. Zur leichteren Lesbarkeit werden die Befehle im Editor automatisch farblich hervorgehoben und in Großbuchstaben konvertiert (Hervorhebung der Syntax).

Ein Befehl ist ein Wort, mit dem eine BASIC Stamp angewiesen werden kann, eine bestimmte Aufgabe auszuführen. Der erste von zwei in diesem Programm benutzten Befehlen ist der **DEBUG** Befehl:

```
DEBUG "Hello, it's me, your BASIC Stamp!"
```

Er weist die BASIC Stamp an, eine Mitteilung über das Verbindungskabel an den PC zu senden.

Der zweite Befehl ist der **END** Befehl:

```
END
```

Mit diesem Befehl schaltet die BASIC Stamp nach Ausführung eines Programms in eine Art Energiesparmodus. In diesem Zustand wartet sie darauf, dass der Resettaster gedrückt oder ein neues Programm über den BASIC Stamp Editor geladen wird. Drücken des Resettaster führt das bereits geladene Programm der BASIC Stamp erneut aus. Nach dem Übertragen eines neuen Programms ist das alte Programm gelöscht und das neue kann sofort ausgeführt werden.

Du bist dran – DEBUG Formatanweisungen und Steuerzeichen

Eine **DEBUG** Formatanweisung ist ein Codewort, das die Darstellung von Mitteilungen der BASIC Stamp im Debug Terminal beeinflusst. **DEC** ist ein Beispiel für eine Formatanweisung, die im Debug Terminal Dezimalwerte anzeigt. Ein Beispiel für ein Steuerzeichen ist **CR (carriage return = Zeilenrücklauf)**; es wird benutzt, um im Debug Terminal einen Zeilenvorschub auszuführen. Text und Zahlen werden nach einem **CR** am Anfang einer neuen Zeile ausgegeben. In Programme lassen sich zusätzliche **DEBUG** Befehle mit Formatanweisungen und Steuerzeichen einfügen. Hier ein Beispiel:

- √ Über *Datei* - *Speichern unter* sicher zunächst dein Programm unter einem neuen Namen. Gib ihm den neuen Namen *FirstProgramYourTurn.bs2*.
- √ Änder die Kommentarzeilen im Programmkopf wie folgt:

```
' What's a Microcontroller - FirstProgramYourTurn.bs2
' BASIC Stamp sends messages to Debug Terminal.
```

- √ Füge die folgenden drei Zeilen zwischen dem ersten **DEBUG** Befehl und dem **END** Befehl ein:

```
DEBUG CR, "What's 7 X 11?"
DEBUG CR, "The answer is: "
DEBUG DEC 7 * 11
```

√ Über *Datei – Speichern* sicherst du die Änderungen.

Das Programm sollte jetzt so wie Abbildung 1-42 aussehen.

√ Führe das veränderte Programm aus. Wähle dazu entweder den Befehl *Run* aus dem Menü *Run* (siehe Abbildung 1-39) oder klicke auf *Run*, wie in Abbildung 1-41 gezeigt.

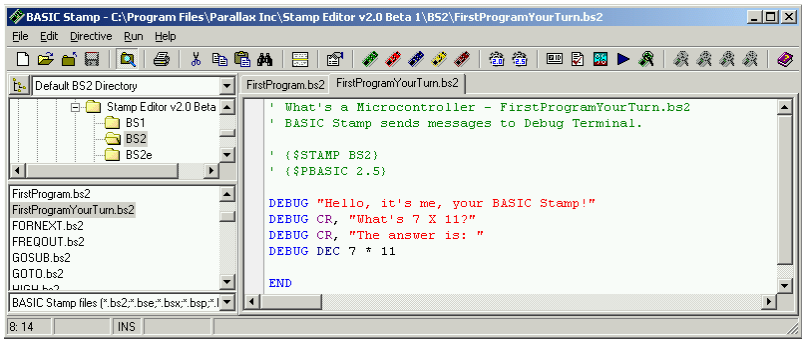


Abbildung 1-42
Geändertes Programm *FirstProgram*

Vergleiche das eingegebene Programm mit diesem Beispiel.

Wo befindet sich das Debug Terminal? Manchmal wird das Debug Terminal vom BASIC Stamp Editor Fenster verdeckt. Es lässt sich über das Menü *Run* zurückholen wie links in Abbildung 1-43 dargestellt, über die *Debug Terminal 1* Kurzwahltaste (Abbildung rechts) oder über die F12-Taste der Tastatur.

Abbildung 1-43

Debug Terminal 1 in den Vordergrund

Über das Menü (links) und über die Kurzwahltaste (rechts.

Das *Debug Terminal* sollte jetzt etwa so aussehen wie in Abbildung 1-44.

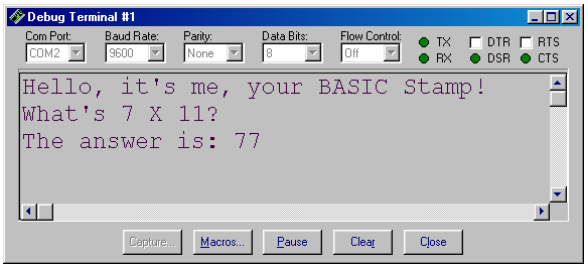


Abbildung 1-44
Ausgabe auf dem Debug Terminal Beispielprogramm: FirstProgram.bs2

Überprüfe, ob beim Neustart des Programms die erwarteten Ergebnisse erscheinen.

ÜBUNG 5: HILFE ANFORDERN

Wir haben in der vorherigen Übung zwei PBASIC Befehle: **`DEBUG`** und **`END`** kennen gelernt. Weitere Hinweise zu diesen Befehlen und wie man sie einsetzt findet man im Hauptmenü des BASIC Stamp Editors unter *Help* und im BASIC Stamp Manual. In dieser Übung wird beispielhaft gezeigt, wie man Hilfe zum Befehl **`DEBUG`** über die Hilfefunktion des BASIC Stamp Editors und im BASIC Stamp Manual bekommt.

Die Hilfefunktion im BASIC Stamp Editor

√ Klicke auf *Help* aus dem Hauptmenü und wähle *Index* (Abbildung 1-45).

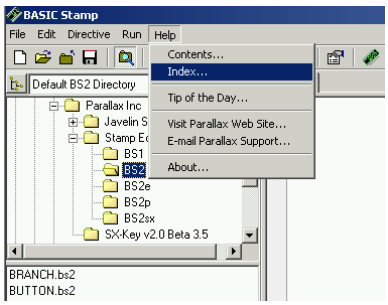


Abbildung 1-45
Wähle *Index* aus dem Menü *Help*

√ Gib in das Feld mit der Bezeichnung *Zu suchendes Schlüsselwort:* DEBUG ein(Abbildung 1-46).

√ Sobald das Wort DEBUG in der darunter befindlichen Liste erscheint, aktiviere *Anzeigen*.

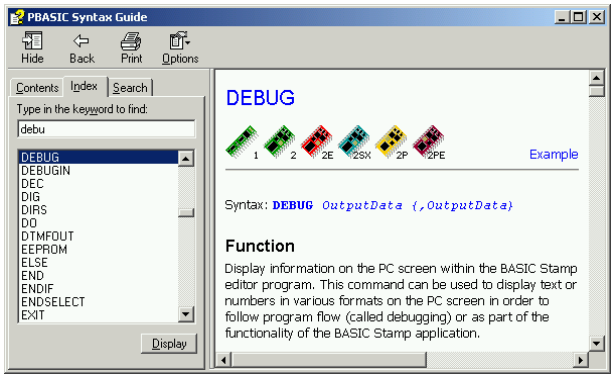


Abbildung 1-46
Über die Hilfefunktion nach Erläuterungen zum DEBUG-Befehl suchen.

Du bist dran

- √ Schaue die Erklärungen zum Befehl **`DEBUG`** an. Beachte, dass es mehrere Erklärungen und Beispielprogramme gibt.
- √ Klicke auf das Register *Inhalt* und suche DEBUG.
- √ Klicke auf das Register *Suchen* und starte den Suchlauf nach dem Wort DEBUG.
- √ Wiederhole diesen Prozess für den Befehl **`END`**.

Das BASIC Stamp Manual

Das BASIC Stamp Manual kann kostenlos von der Parallax Website herunter geladen oder als Buch gekauft werden. Es befindet sich auch auf der Parallax CD.

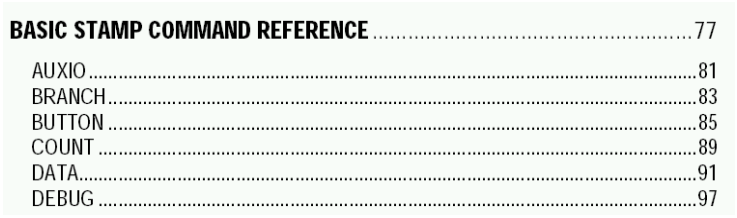
Herunterladen des BASIC Stamp Manuals von der Parallax Website

- √ Gehe im Browser auf www.parallax.com.
- √ Lasse dir über das Menü *Downloads* die Optionen anzeigen.
- √ Klicke auf *Documentation* .
- √ Auf der Seite *BASIC Stamp Documentation* befindet sich das *BASIC Stamp Users Manual*.
- √ Klicke auf das Ordner-Symbol *Download*, rechts von der Beschreibung: “BASIC Stamp User’s Manual Version 2.0 (3.2 MB)”.

Nachschlagen im BASIC Stamp Manual auf der Parallax CD

- √ Aktiviere *Documentation*.
- √ Klicke auf + neben dem Ordner BASIC Stamps.
- √ Aktiviere das Symbol BASIC Stamp Manual book.
- √ Klicke auf *View*.

√ Abbildung 1-47 zeigt einen Auszug aus dem Inhaltsverzeichnis des *BASIC Stamp Manual* v2.0, aus dem hervor geht, das Informationen über den Befehl **DEBUG** auf Seite 97 zu finden sind.

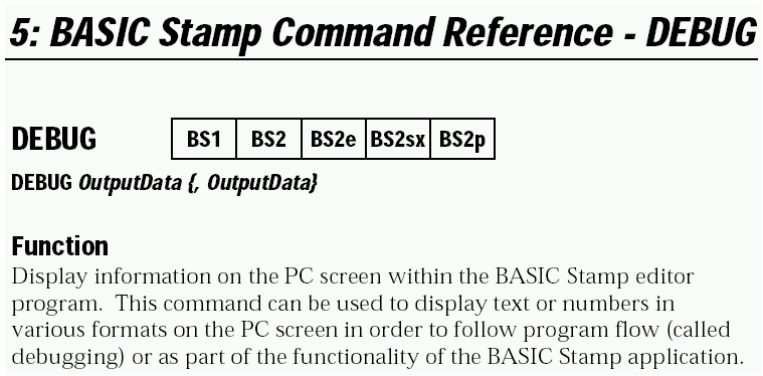

BASIC STAMP COMMAND REFERENCE 77

AUXIO 81
BRANCH 83
BUTTON 85
COUNT 89
DATA 91
DEBUG 97

Abbildung 1-47 Aufsuchen des DEBUG Befehls im Inhaltsverzeichnis

Abbildung 1-48 zeigt einen Auszug der Seite 97 aus dem *BASIC Stamp Manual v2.0*. Der Befehl **DEBUG** wird hier mit Beispielprogrammen, wie man den Befehl einsetzen kann, ausführlich erklärt.

√ Schau dir alle Hinweise zum **DEBUG** Befehl im *BASIC Stamp Manual* an.
√ Wie viele Beispielprogramme gibt es zu diesem Befehl?



5: BASIC Stamp Command Reference - DEBUG

DEBUG | BS1 | BS2 | BS2e | BS2sx | BS2p

DEBUG *OutputData {, OutputData}*

Function

Display information on the PC screen within the BASIC Stamp editor program. This command can be used to display text or numbers in various formats on the PC screen in order to follow program flow (called debugging) or as part of the functionality of the BASIC Stamp application.

Abbildung 1-48 Darstellung des Befehls **DEBUG** im BASIC Stamp Manual

Du bist dran

√ Besorge dir Informationen über den Befehl **DEBUG** aus dem Indexverzeichnis des *BASIC Stamp Manuals*.
√ Verschaffe dir über das *BASIC Stamp Manual* Informationen zum Befehl **END**.

ÜBUNG 6: DER ASCII CODE

In Übung 4 haben wir über den Befehl **DEBUG** den Formatierungsbefehl **DEC** kennengelernt, um eine Dezimalzahl im Debug Terminal darzustellen. Wird der Befehl **DEBUG** ohne Formatierungsbefehl **DEC** aufgerufen, interpretiert die BASIC Stamp die Zahl als sogenannten ASCII Code.

Programmieren mit ASCII Code

ASCII ist eine Abkürzung und steht für *American Standard Code for Information Interchange*. Von den meisten Mikrocontrollern und Computern wird dieser Code benutzt, der jeder Computertaste und komplexeren Tastenkombinationen eine bestimmte Zahl zuweist. Die Zahlen von 32 bis 126 gehören zu den Zeichen und Symbolen, die von der BASIC Stamp im Debug Terminal dargestellt werden können. Im folgenden Programm wird mit Hilfe des ASCII-Codes der Begriff „BASIC Stamp 2“ im Debug Terminal angezeigt.

Programmbeispiel – ASCIIName.bs2

√ Gib das Programm ASCIIName.bs2 ein und führe es anschließend aus.

Denke bitte daran, die Compilerdirektiven über die Symbole aus der Werkzeugleiste einzugeben!

'{$STAMP BS2} - Use the diagonal green electronic chip icon.
'{$PBASIC 2.5} - Use the gear icon labeled 2.5.

You can see a picture of these icons again on page 22.

```
'What's a Microcontroller - ASCIIName.bs2
'Use ASCII code in a DEBUG command to display the words BASIC Stamp 2.

'{$STAMP BS2}
'{$PBASIC 2.5}

DEBUG 66,65,83,73,67,32,83,116,97,109,112,32,50
```

```
END
```

Wie arbeitet das Programm ASCIIName.bs2?

Zu jeder Zahl in einem **DEBUG** Befehl gehört ein ASCII-Code Symbol, das im Debug Terminal angezeigt wird.

```
DEBUG 66,65,83,73,67,32,83,116,97,109,112,32,50
```

66 ist der ASCII Code für das große B, 65 für das große A und so weiter. 32 steht für Leerzeichen. Beachte, dass hinter dem Befehl **DEBUG** die Codezahlen durch Komma getrennt werden. Auf diese Art können sie mit nur einem Befehl einzeln abgearbeitet werden; dies ist sicherlich platzsparender und übersichtlicher als für jeden ASCII-Code eine **DEBUG** –Zeile einzugeben.

Du bist dran – Entdecke den ASCII Code

- √ Speicher ASCIIName.bs2 unter dem neuen Namen ASCIIRandom.bs2
- √ Wähle nach dem Zufallsprinzip 12 Zahlen zwischen 32 und 127 aus.
- √ Ersetze die ASCII-Code Zahlen des Programms durch eigene Zufallszahlen.
- √ Führe das veränderte Programm aus und beobachte was passiert!

Im Anhang A des *BASIC Stamp Manuals* findest du eine ASCII-Code Tabelle mit den entsprechenden Symbolen. Versuche, deinen eigenen Namen im ASCII-Code anzugeben.

- √ Speicher ASCIIRandom.bs2 unter dem neuen Namen YourASCIIName.bs2
- √ Schlage die ASCII-Tabelle im BASIC Stamp Manual auf.
- √ Gib deinen Namen in das Programm ein.
- √ Führe das Programm aus und überprüfe, ob du ihn richtig buchstabiert hast.
- √ Wenn ja, hast du alles richtig gemacht. Speicher das Programm jetzt ab!

ÜBUNG 7: DIE ARBEIT BEENDEN

Gewöhne dich daran, die BASIC Stamp und das Board of Education (oder HomeWork Board) nach Beendigung der Arbeit von der Spannungsquelle zu trennen. Erstens wird es die Batterie mit einer längeren Lebensdauer danken. Zweitens sollten Entwicklungsschaltungen – wie wir sie demnächst aufbauen werden - niemals unbeaufsichtigt gelassen werden.

Versuchsschaltungen sollten, wenn sie mit einer Spannungsquelle verbunden sind, niemals unbeaufsichtigt gelassen werden. **Unterbrich die Spannungsversorgung** zu deinem Board of Education oder HomeWork Board bevor du den Arbeitsplatz verlässt – auch wenn es nur eine Minute sein sollte.

Unterbrechen der Spannungsversorgung

Das Unterbrechen der Spannungsversorgung ist beim Board of Education Rev C sehr einfach. Schiebe dazu den 3-fach Schiebeschalter einfach nach links in Position 0 (siehe Abbildung 1-49).

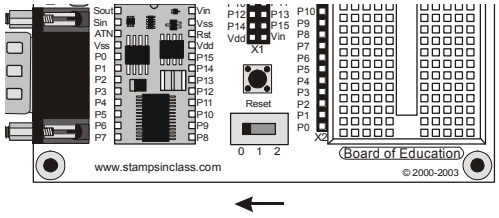


Abbildung 1-49
Unterbrechen der Spannungsversorgung

Board of Education Rev C

Ziehe auf keinen Fall das BASIC Stamp Modul aus seiner Fassung! Jedesmal, wenn die BASIC Stamp aus der IC-Fassung auf dem Board of Education herausgenommen und anschließend wieder hinein gesteckt wird, riskiert man Beschädigungen.

Auch beim BASIC Stamp HomeWork Board ist die Unterbrechung der Spannungversorgung einfach. Löse dazu einfach die Batterie vom Clip, so wie in Abbildung 1-50 dargestellt.

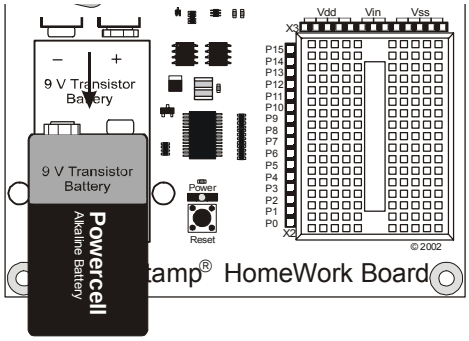


Abbildung 1-50
Löse die Batterie von den Clips auf dem HomeWork Board

Das Board of Education Rev B verfügt über keinen 3-fach Schiebeschalter. Ziehe den Netzstecker, um das Board von der Spannungsversorgung zu trennen oder entferne die Batterie vom Clip.

Du bist dran

√ Unterbreche die Spannungsversorgung deines Boards.

ZUSAMMENFASSUNG

In diesem Kapitel hast du gelernt:

- in welchen Geräten Mikrocontroller eingesetzt werden.
- was man unter einem BASIC Stamp Modul versteht.
- das man interessante Entwicklungen mit BASIC Stamp Modulen aufbauen kann.
- wo der BASIC Stamp Editor, der in diesem Buch in nahezu allen Übungen eingesetzt wird, kostenlos heruntergeladen werden kann.
- wie man den BASIC Stamp Editor auf einem Rechner installiert.
- das *BASIC Stamp Modul*, das *Board of Education* und das *HomeWork Board* voneinander zu unterscheiden.
- wie die BASIC Stamp Hardware eingerichtet wird.
- wie die Soft- und Hardware ausgetestet werden kann.
- wie man ein PBASIC Programm schreibt und ausführt.
- die Befehle **`DEBUG`** und **`END`** zu benutzen.
- das Steuerzeichen **`CR`** und die Formatanweisung **`DEC`** in einem Programm zu nutzen.
- wie man die Hilfefunktion des BASIC Stamp Editors aufruft und das BASIC Stamp Manual nutzt.
- was unter einem ASCII Code zu verstehen ist.
- wie man die Spannungsversorgung des *Board of Education* oder des *HomeWork Boards* nach beendeter Arbeit unterbricht.

Verständnisfragen

1. Was versteht man unter einem Mikrocontroller?
2. Ist das BASIC Stamp Modul ein Mikrocontroller oder enthält es einen?
3. Woran lässt sich schnell erkennen, ob ein technisches Gerät wie z.B. ein Wecker oder ein Mobilphone einen Mikrocontroller enthält?
4. Worauf weist das Apostroph am Anfang einer Programmzeile in einem PBASIC Programm hin?
5. Welche PBASIC Befehle hast du in diesem Kapitel kennen gelernt?
6. Nehmen wir an, du möchtest dein BASIC Stamp Projekt, an dem du gerade arbeitest, für eine kurze Zeit unterbrechen oder erst am nächsten Tag fortsetzen. Woran solltest du vor dem Verlassen des Arbeitsplatzes stets denken?

Aufgaben

1. Erkläre die Wirkung des Stern-Symbols in diesem Befehl:

```
DEBUG DEC 7 * 11
```

2. Bestimme, was das *Debug Terminal* bei Ausführung des unten stehenden Befehls anzeigen würde:

```
DEBUG DEC 7 + 11
```

3. Es gibt ein Problem mit den beiden folgenden Befehlen. Wird der Befehl ausgeführt, gibt das Programm die Ergebnisse in einer Zeile hintereinander aus. Änder das Programm so ab, dass die Ergebnisse jeweils in einer Zeile auf dem *Debug Terminal* ausgegeben werden.

```
DEBUG DEC 7 * 11
DEBUG DEC 7 + 11
```

Projekte

1. Gib mit Hilfe des Befehls **DEBUG** die Lösung folgender Aufgabe auf dem Debug Terminal aus: 1 + 2 + 3 + 4.
2. Welche Zeilen des Programms FirstProgramYourTurn.bs2 können gelöscht werden, wenn du die unten dargestellte Programmzeile vor dem Befehl **END** in das Programm einfügst? Bevor du das Programm umschreibst, gib vorher eine Vermutung über sein Verhalten bezüglich deiner vorgeschlagenen Lösung ab. Speicher das Programm FirstProgramYourTurn.bs2 unter einem neuen Namen, wie z.B. FirstProgramCh01Project05.bs2 und füge anschließend deine Änderungen in das Programm ein.

```
DEBUG "What's 7 X 11?", CR, "The answer is: ", DEC 7 * 11
```

Lösungen

Q1. Ein Mikrocontroller ist eine Art Minicomputer, der in vielen elektronischen Geräten zu finden ist.
Q2. Das BASIC Stamp Modul enthält den Mikrocontroller PIC16C57.
Q3. Geräte mit Knöpfen und einem digitalen Display lassen vermuten, dass dort Mikrocontroller eingebaut worden sind.
Q4. Eine Kommentarzeile.
Q5. **DEBUG** und **END**
Q6. Löse die Spannungsversorgung vom BASIC Stamp Projekt.
E1. Die Zahlen 7 und 11 werden miteinander multipliziert, das Ergebnis ist 77. Das Stern-Symbol ist ein Multiplikationsoperator.
E2. Das Debug Terminal zeigt 18 an.

E3. Das Problem wird dadurch behoben, dass man ein CR (Carriage Return = Zeilenrücklauf) als Kontrollzeichen in das Programm einfügt.

```
DEBUG DEC 7 * 11
DEBUG CR, DEC 7 + 11
```

P1. Das Programm gibt eine Lösung der Aufgabe 1+2+3+4 aus.

```
'{$STAMP BS2}
'{$PBASIC 2.5}

DEBUG "What's 1+2+3+4?"
DEBUG CR, "The answer is: "
DEBUG DEC 1+2+3+4

END
```

P2. Die drei letzten **DEBUG** Zeilen können gelöscht werden. Es muss zusätzlich ein **CR** nach der "Hello" Mitteilung eingegeben werden.

```
' What's a Microcontroller - FirstProgramYourTurn.bs2
' BASIC Stamp sends message to Debug Terminal.

' {$STAMP BS2}
' {$PBASIC 2.5}

DEBUG "Hello, it's me, your BASIC Stamp!", CR

DEBUG "What's 7 X 11?", CR, "The answer is: ", DEC 7 * 11

END
```

Die Ausgabe im Debug Terminal lautet::

```
Hello, it's me, your BASIC Stamp!
What's 7 X 11?
The answer is: 77
```

Diese Ausgabe ist identisch mit dem vorherigen Programmcode. In diesem Beispiel wurde das Komma benutzt, um eine Vielzahl von Informationen mit nur einem **DEBUG** Befehl aufzurufen.

Weiterführende Bücher

In diesem Kapitel haben wir gezeigt, wie man an eine Kopie des BASIC Stamp Editors entweder aus dem Internet oder der Parallax CD kommt. Sowohl im Internet als auch auf der Parallax CD findest du den Text dieses Buches und das *BASIC Stamp Manual*. Alle Texte können auch in gebundener Form bei Parallax bestellt werden.

***"BASIC Stamp Manual"*, Benutzerhandbuch, Version 2.0c, Parallax Inc., 2000**

Im *BASIC Stamp Manual* erfährst du mehr über die Befehle **`DEBUG`** und **`END;`** benutze dazu das Inhaltsverzeichnis. Im Manual sind viele Beispiele und Anwendungen abgedruckt, die du direkt ausprobieren kannst.

Kapitel2: Leuchtdioden ansteuern

2

KONTROLLLEUCHTEN

Kontrollleuchten kommen in so vielen Geräten vor, dass wir sie bewusst häufig gar nicht mehr wahrnehmen. In Abbildung 2-1 sind drei Kontrollleuchten eines Laserprinters dargestellt. Je nach dem, welche der Leuchtanzeigen bei einem Drucker aufleuchten, erkennen wir sofort, ob alles in Ordnung ist, der Toner gewechselt werden muss oder ein Papierstau vorliegt. Beispiele für Geräte mit Kontrollleuchten lassen sich leicht finden: Autoradios, Fernseher, Diskettenlaufwerke, Drucker, Kontrollpaneele von Alarmanlagen und Mischpulte.

Abbildung 2-1
Kontrollleuchten

Eine Leuchtanzeige lässt sich auf einfache Weise ein- und ausschalten, indem man die Spannungsversorgung zum Leuchtmittel unterbricht oder wieder herstellt. Manchmal werden Leuchtanzeigen direkt mit einer Batterie oder Spannungquelle verbunden, wie zum Beispiel die Kontrollleuchte des *Board of Education*. Eine andere Methode ist, die Leuchtanzeigen über einen Mikrocontroller ein- bzw. auszuschalten. Auf diese Weise lassen sich Betriebszustände eines Gerätes anzeigen.

EINE LEUCHTDIODE (LED) EINSCHALTEN

Die meisten Leuchtanzeigen sind heute mit sogenannten Leuchtdioden bestückt - kurz LED (light emitting diode) genannt. Eine LED zusammen mit einer Spannungsquelle

(ein LED Schaltkreis) genügt, um die Diode zum Leuchten zu bringen. Unterbricht man die Leitung zur Spannungsquelle, erlischt sie.

Verbindet man einen LED Schaltkreis mit einer BASIC Stamp, lässt sich die BASIC Stamp so programmieren, dass sie die Spannungsversorgung der LED unterbricht oder einschaltet. Ein solches Vorgehen ist sicherlich einfacher als jedesmal die Steckverbindung von Hand zu lösen oder die Batterie aus der Halterung zu ziehen. Die BASIC Stamp lässt sich aber auch für kompliziertere Lichtspiele programmieren. Einige Beispiele sind:

- In einem LED Schaltkreis wird die LED unterschiedlich lang ein- bzw. ausgeschaltet.
- In einem LED Schaltkreis wird die LED zum Beispiel nur 10 mal ein- und ausgeschaltet.
- Es werden mehrere LED Schaltkreise überwacht und gesteuert.
- In einem LED Schaltkreis werden die Farben einer DUO-LED angesteuert.

ÜBUNG 1: AUFBAU UND TEST EINER LED SCHALTUNG

Bevor elektronische Bauteile in eine größere Schaltung eingebaut werden, müssen sie einzeln getestet werden. In dieser Übung konzentrieren wir uns auf den Aufbau und das Austesten von zwei unterschiedlichen LED Schaltkreisen. Mit der ersten Schaltung wird eine LED zum Leuchten gebracht, in einer zweiten Schaltung soll die LED nicht leuchten. In der nachfolgenden Übung wird der erste LED Schaltkreis mit der BASIC Stamp verbunden, und das Ein- und Ausschalten der Diode durch ein kleines Programm, das wir erst schreiben und dann in den Mikrocontroller laden, gesteuert. Dadurch, dass vorher die LED-Schaltungen auf Funktionstüchtigkeit geprüft wurden, können wir sicher sein, dass sie auch funktionieren, wenn sie mit der BASIC Stamp verbunden werden.

Widerstände gibt es überall

Unter einem Widerstand verstehen wir ein elektronisches Bauteil, dass den elektrischen Stromfluss hemmt. Je stärker ein Stromfluss gehemmt wird, um so größer ist der Widerstandswert des elektronischen Bauteils. Der Widerstandswert wird in der Einheit Ohm (abgekürzt mit dem griechischen Buchstaben Ω) angegeben. In dieser Übung arbeiten wir mit einem Widerstand von 470 Ω (siehe Abbildung 2-2). Ein Widerstand besteht aus einem Keramikträger, dem eigentlichen Widerstand, und zwei Anschlussdrähten, die rechts und links angebracht sind. In den meisten Schaltungsplänen werden Widerstände über ihre entsprechenden Schaltungssymbole dargestellt. In diesem

Buch werden ausschließlich Schaltungssymbole nach der US-Norm benutzt, die nicht immer der Deutschen Industrie Norm (DIN) entsprechen. In Abbildung 2-2 siehst du links das Schaltsymbol und rechts eine Bauteilzeichnung.

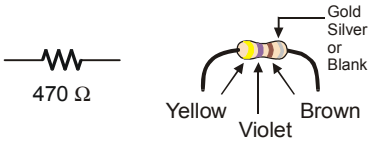


Abbildung 2-2
470 Ω Widerstand

Schaltungssymbol (links)
Bauteil (rechts)

Die meisten Widerstände haben aufgedruckte Farbringe, die einem verraten, welchen Widerstandswert das Bauteil hat. Je nach Ausführung sind es vier oder fünf Farbringe; häufig ist der äußerste Farbring silber- oder goldfarben. Ein 470 Ω Widerstand trägt zum Beispiel die Farbcodierung gelb-violett-braun.

Der äußerste Farbring sagt etwas aus über die Genauigkeit oder Fehlertoleranz des Widerstandswertes. Die Fehlertoleranz wird in Prozent (%) angegeben und beschreibt, um wie viel Prozent der angegebene Widerstandswert von seinem tatsächlichen Wert abweichen darf. Der äußerste Ring trägt bei den meisten Widerständen die Farbe Gold (5%) oder Silber (10%). Das heißt, ein 100 Ω Widerstand mit goldenem Farbring, kann einen realen Widerstandswert zwischen 95 Ω und 105 Ω haben. Der Fehlertoleranzwert ist in den Übungen dieses Buches vernachlässigbar; wichtig sind nur die angegebenen Widerstandswerte.

Jeder Farbring auf einem Widerstand symbolisiert eine Zahl, die Position des Farbringes gibt den Stellenwert dieser Zahl an. Die zu den Farben korrespondierenden Zahlenwerte findest du in Table 2-1. Abbildung 2-3 zeigt, wie über die Farbringe mit Hilfe der Tabelle ein Widerstandswert bestimmt wird.

Table 2-1: Widerstand Farbcode

Zahl	Farbe
0	Schwarz
1	Braun
2	Rot
3	Orange
4	Gelb
5	Grün
6	Blau
7	Violett
8	Grau
9	Weiß

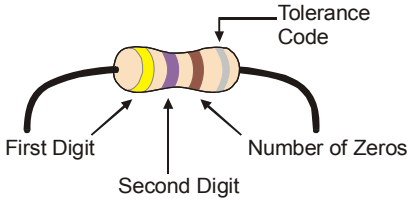


Abbildung 2-3 Widerstand Farbcode

In dem folgenden Beispiel erfährst du, wie man mit Hilfe der Table 2-1 und Abbildung 2-3 feststellen kann, dass der vorliegende Widerstand mit der Farbcodierung gelb-violett-braun einen Widerstandswert von 470 Ω hat:

- Lege den Widerstand so vor dich hin, dass der Toleranzring (gold oder silber) rechts außen liegt.
- Der erste Farbstreifen (ganz links) zeigt gelb an, dass entspricht einer 4.
- Der zweite Farbstreifen ist violett, dass entspricht einer 7.
- Der braune Streifen steht für eine 1. Als letzter Farbstreifen gibt diese Zahl an, wie viele Nullen an die beiden ersten Ziffern anzuhängen sind; in diesem Beispiel eine Null, da brauner Ring.

Gelb-Violett-Braun = 4-7-0.

Erste Schritte mit einer LED

Eine Diode arbeitet wie ein Ventil, das den Strom nur in einer Richtung durchlässt. Eine Leuchtdiode wandelt elektrische Energie in Lichtenergie um, dabei gibt die Farbe des Leuchtdiodenkörpers an, in welcher Farbe die LED leuchten wird. Wenn du eine Leuchtdiode in einen Schaltkreis einbaust, muss du unbedingt auf die richtige Anschlusspolung achten, damit der Strom durch die Diode fließen und sie zum Leuchten bringen kann. Wie das geschieht, erfährst du in diesem Abschnitt.

In Abbildung 2-4 wird das Schaltsymbol und die Bauteilzeichnung dargestellt. Eine LED verfügt über zwei Anschlüsse, die wir auch Elektroden nennen. Die längere Elektrode wird als Anode, die kürzere als Kathode bezeichnet. In der Bauteilzeichnung wird die Anode mit einem (+)-Zeichen markiert und ist dem längeren Beinchen des Bauteils zugeordnet. Beim Schaltsymbol befindet sich die Anode an der Basis, die Kathode an der Spitze des Dreiecks. In der folgenden Übung wird eine LED in einen Schaltkreis eingebaut.

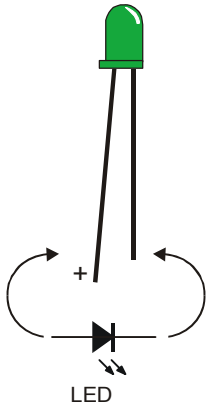


Abbildung 2-4
LED Bauteilzeichnung und Schaltsymbol

Bauteilzeichnung *(oben) und Schaltsymbol (unten).*

Die Anode, das ist das längere Anschlussbeinchen des Halbleiters, wird an den +-Pol der Energiequelle angeschlossen.

Wenn du die Schaltung aufbaust, musst du unbedingt auf die richtige Anschlussbelegung achten. Der Bauteilzeichnung kannst du entnehmen, dass die Beinchen einer LED unterschiedlich lang sind. Die längere Elektrode ist mit der Anode, die kürzere mit der Kathode einer LED verbunden. Bei genauerem Hinsehen erkennt man, dass der runde Leuchtkörper der LED auf der Seite der kürzeren Elektrode abgeflacht ist. Damit kennen wir zwei Möglichkeiten, die Anode einer Diode sicher zu bestimmen.

Bauteile der LED Testschaltung

1 LED grün
1 Widerstand 470 Ω (gelb-violett-braun)

Bauteile identifizieren: Zusätzlich zu den Bauteilezeichnungen aus Abbildung 2-2 und Abbildung 2-4, kannst du auch über das Photo auf der letzten Seite dieses Buches die Bauteile für diese und die folgenden Übungen identifizieren. Weitere Informationen zu den Bauteilen auf dem Photo erhälst du im Anhang B: Ausstattung und Bauteile.

Aufbau einer LED Testschaltung

Für den Aufbau eines Schaltkreises benutzen wir ein weißes Steckbrett mit schwarzen Buchsenleisten, die oben und links angebracht sind (siehe Abbildung 2-5). Die obere Buchsenleiste (beschriftet mit: Vdd, Vin, und Vss) ist die Anschlussleiste für die Spannungsversorgung, während die Anschlüsse der linken Buchsenleiste (beschriftet mit: P0, P1 ... P15) mit den Eingabe/Ausgabe Pins des BASIC Stamp Moduls verbunden sind. Mit Hilfe des weißen Steckbretts kann schnell und sicher ein lötfreier Schaltungsaufbau realisiert werden.

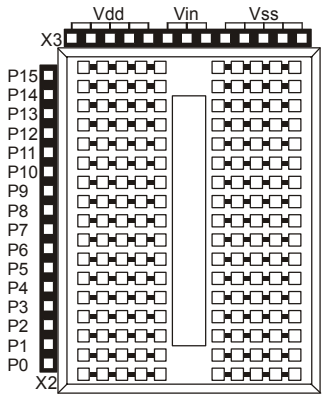


Abbildung 2-5
Prototypfeld

Spannungsversorgung (schwarze Buchsenleiste oben), Ein-/Ausgabepins (schwarze Buchsenleiste links) und weißes Steckbrett für lötfreien Schaltungsaufbau

Eingabe-/Ausgabestifte werden kurz als I/O Pins bezeichnet. Nachdem ein Schaltkreis mit einem oder mehreren I/O Pins verbunden wurde, lässt sich die BASIC Stamp so programmieren, dass der Schaltkreis Signale empfängt (INPUT) oder das an den Schaltkreis EIN/AUS Signale gesendet werden (OUTPUT). Wir werden das in der nächsten Übung probieren.

Abbildung 2-6 zeigt einen Schaltkreis und das Bild des Schaltungsaufbaus auf dem Steckbrett. Bei genauerem Hinsehen erkennt man, dass das Steckbrett über Zeilen mit je fünf verbundenen Buchsen verfügt. In jeder Zeile lassen sich bis zu fünf Anschlussdrähte unterbringen.

Bau die Schaltung mit einer LED und einem Widerstand so auf wie in Abbildung 2-6 (unten) dargestellt. Halte dabei die folgende Reihenfolge ein:

√ Trenne das *Board of Education* oder das *HomeWork Board* von der Spannungsquelle.

√ Bau die Schaltung genau so auf wie in Abbildung 2-4 dargestellt. Achte unbedingt auf die richtige Verbindung der Anschlussdrähte der LED. Dazu musst du erst einmal herausfinden, welche Elektrode mit der Anode verbunden ist.

Bei einer LED ist die Anschlussbelegung wichtig, bei einem Widerstand nicht. Wird eine LED falsch beschaltet (Vertauschung von + und – Pol), dann wird sie in einem Schaltkreis nicht leuchten. Bei einem Widerstand müssen wir uns um die Polung der Beschaltung keine Gedanken machen.

√ Verbinde dein *Board of Education* oder *HomeWork Board* mit der Spannungsquelle.
√ Überprüfe, ob die grüne LED aufleuchtet.

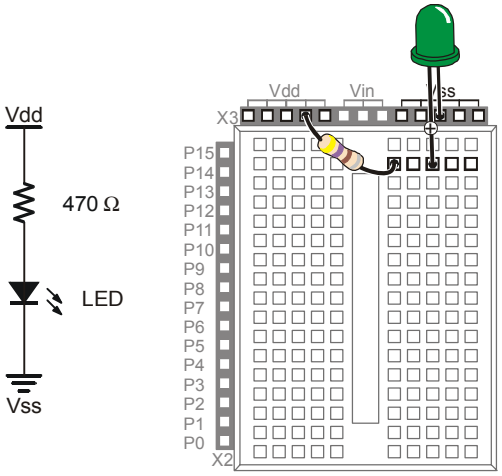


Abbildung 2-6
LED leuchtet

Schaltplan (links) und Steckbrett-Verkabelung (rechts)

Sollte die grüne LED nicht aufleuchten, obwohl die Spannungsversorgung angelegt ist:

√ dann überprüfe, indem du direkt von oben auf die LED schaust, ob ein Leuchten erkennbar ist oder wenn der Raum stark beleuchtet ist
√ schatte die LED mit deinen Händen etwas ab und beobachte sie dann.

Solltest du damit kein Glück haben, gehe wie folgt vor:

- √ Überprüfe mehrfach, ob Kathode und Anode der LED richtig angeschlossen sind. Wenn nicht, vertausche die Anschlussbelegung. Die LED sollte jetzt leuchten.
- √ Überprüfe genau, ob du die Schaltung genau so wie in Abbildung 2-6 dargestellt, aufgebaut hast.
- √ Arbeitest du mit Bauteilen, die vor dir schon andere benutzt haben, dann kann die LED defekt sein - tausche sie gegen eine andere aus.
- √ Wenn möglich, frage deinen Lehrer.

Der Fehler lässt sich nicht beseitigen? Wenn du keine direkte Hilfe z.B. von Freunden oder Lehrkräften bekommen kannst, hole dir Hilfe über die englischsprachigen *Stamps in Class* Diskussionsforen im Internet. Auf den ersten Seiten dieses Buches findest du Informationen über Internetzugänge zu den o.g. Diskussionsforen. Sollte auch dies nicht zum Ziel führen, wende dich an den technischen Support von Parallax, Inc. unter: www.parallax.com.

Arbeitsweise der LED Testschaltung

Wird eine LED so, wie in Abbildung 2-7 gezeigt, beschaltet, beginnt sie zu leuchten. Die Anode ist über einen Widerstand mit dem + Pol einer Batterie verbunden, die Kathode direkt mit dem – Pol. Die negativen Ladungsteilchen am Minuspol der Batterie stoßen sich gegenseitig ab und werden gleichzeitig vom + Pol der Batterie (Mangel an Elektronen) angezogen; sie bewegen sich – nur gebremst durch den Widerstand der Leitungsverbindung – durch die LED zum + Pol. Diese Bewegung bezeichnen wir als elektrischen Strom. Die LED beginnt zu leuchten.

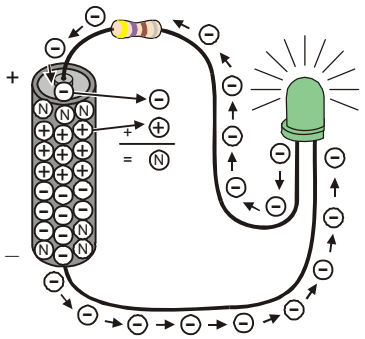


Abbildung 2-7
Elektronenfluss in einem Schaltkreis in dem die „LED leuchtet“

Die Kreise mit den Minuszeichen symbolisieren den Fluss der negativen Ladungsteilchen vom Minuspol zum Pluspol einer Batterie.

Chemische Wechselwirkungen innerhalb der Batterie sind für den Stromfluss verantwortlich. Am negativen Pol einer Batterie gibt es einen Überschuss an Elektronen (in Abbildung 2-7 mit Minuszeichen dargestellt). Entsprechend herrscht am Pluspol der Batterie ein Mangel an Elektronen (dargestellt durch ein + Zeichen). Elektronen, die sich durch den Draht von einem Pol zum anderen bewegen können, werden als sogenannte freie Elektronen bezeichnet; sie werden auch mit einem – Zeichen symbolisiert. Moleküle, die über keine freien Ladungsträger verfügen, bezeichnen wir als **neutral** (dargestellt durch ein N). Gelangt ein Elektron zum Pluspol einer Batterie (dort fehlt den Molekülen ein Elektron), wird das positive Molekül durch Einbindung des Elektrons neutral.

In Abbildung 2-8 wird der Stromfluss durch einen Schaltkreis mit LED dargestellt. Der Pfeil zeigt die sogenannte **technische Stromrichtung** an: sie gibt die Richtung der Ladungsbewegung vom Plus- zum Minuspol einer Energiequelle an. Tatsächlich bewegen sich negative Ladungsteilchen auf Grund ihrer elektrischen Eigenschaften immer vom Minus- zum Pluspol. Bezieht man sich auf diese Sichtweise der Ladungsbewegung, dann spricht man von der sogenannten **physikalischen Stromrichtung**. Bei Schaltplänen mit eingezeichneten Stromverläufen musst du dir vorher klar machen, ob die technische oder physikalische Stromrichtung angegeben wurde.
Zwischen den Polen der Batterie existiert ein Ladungsdruck, den wir auch als Spannung bezeichnen. Die + und – Zeichen auf einer Batterie geben an, auf welcher Seite negativer Ladungsüberschuss (- Pol) bzw. Ladungsmangel (+ Pol) herrscht.

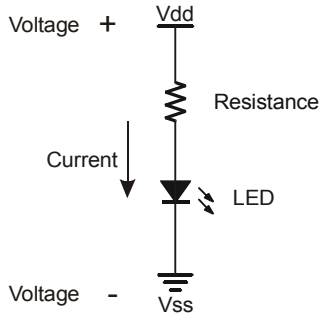


Abbildung 2-8
Schaltkreis LED-an

Der elektrische Strom fließt vom + zum – Pol der Energiequelle (technische Stromrichtung).

In einem Schaltplan (Abbildung 2-8) wird der Zusammenschluss verschiedener elektrischer Schaltungsteile dargestellt.

Anhang F: : In diesem Anhang findest du viele Begriffserklärungen und eine Übung, die zeigt, wie man Strom-, Spannungs- und Widerstandsmessungen vornimmt.

Du bist dran – Modifikation der LED Testschaltung

In der folgenden Übung wirst du die BASIC Stamp so programmieren, dass sie die LED in einem vorgegebenen Rhythmus an- und abschaltet. Das gelingt durch einen elektronischen Schalter - die BASIC Stamp – der ständig zwischen den Anschlüssen Vdd und Vss umschaltet. Baue die Schaltung nach Abbildung 2-9 auf und überprüfe, ob die LED abschaltet, wenn der Anschluss des Widerstandes von der Vdd- zur Vss-Buchse gelegt wird.

- √ Trenne das Board of Education oder HomeWork Board von der Energiequelle.
- √ Ziehe den Anschlussdraht des Widerstandes aus der Vdd Buchse und stecke ihn in die Vss Buchse (siehe Abbildung 2-9).
- √ Verbinde die Energiequelle wieder mit dem Board of Education oder Home-Work Board.
- √ Vergewissere dich, dass die grüne **LED nicht leuchtet**.

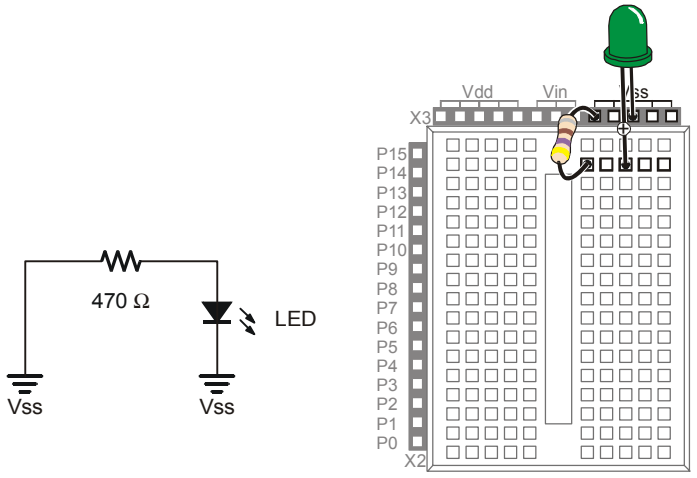


Abbildung 2-9
Schaltkreis LED Aus

Schaltplan (links) und Verdrahtungsplan (rechts).

ÜBUNG 2: DIE BASIC STAMP ALS EIN/AUS SCHALTER

In Übung 1 haben wir zwei unterschiedliche Schaltungen aufgebaut und ausgetestet. In der einen wurde die LED so in einen Schaltkreis eingebaut, dass sie leuchtete, sobald die Spannungsquelle angelegt wurde, während sie in der zweiten Schaltung abschaltete. Abbildung 2-10 zeigt, wie man mit einer BASIC Stamp die gleichen Ergebnisse erzielen kann, wenn der LED Schaltkreis mit einem der I/O Anschlüsse des Steckbretts verbunden

wird. In dieser Übung wird ein LED Schaltkreis mit einer BASIC Stamp verbunden und so programmiert, dass die LED mit unterschiedlichen Geschwindigkeiten ein- und ausschaltet.

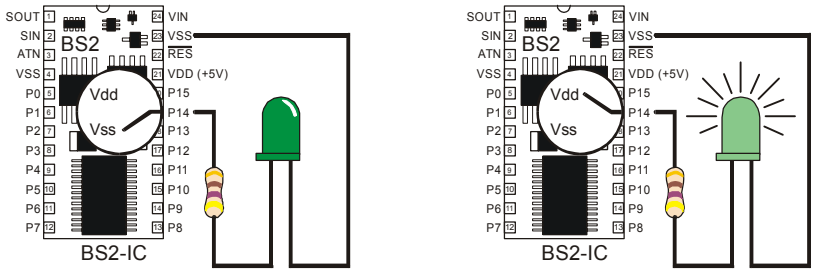


Abbildung 2-10
BASIC Stamp als Schalter

Die BASIC Stamp wird so programmiert, dass der LED Schaltkreis nacheinander mit der Buchse Vdd und Vss verbunden wird.

Das Umschalten der Verbindung von Vdd zu Vss und zurück mit Hilfe der BASIC Stamp erfolgt ohne vorheriges Abschalten der Energiequelle, gleichzeitig schaltet der Mikrocontroller wesentlich schneller um, als es der Mensch kann. Dies sind zwei gravierende Vorteile, die für den Einsatz der BASIC Stamp sprechen.

Bauteile der LED Testschaltung

Siehe Schaltung aus Übung 1.

Verbindung von LED Schaltkreis mit BASIC Stamp

Der LED Schaltkreis in Abbildung 2-11 unterscheidet sich kaum von den Schaltungen, die wir bisher aufgebaut haben. Neu ist, dass der Anschluss des Widerstandes, der bisher zwischen den Buchsen Vdd und Vss mit der Hand umgepolt werden musste, jetzt fest mit einer BASIC Stamp I/O Buchse verbunden wird.

- √ Trenne das Board of Education oder HomeWork Board von der Energiequelle.
- √ Baue die Schaltung ausÜbung 1 so um, dass sie der in Abbildung 2-11 entspricht.

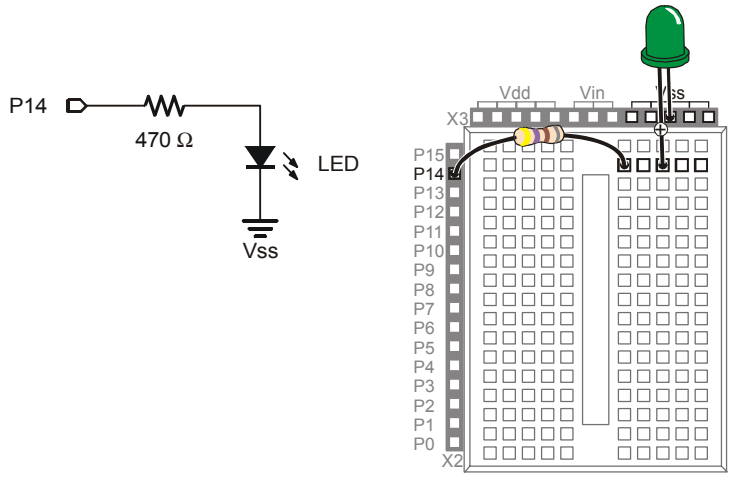


Abbildung 2-11
LED Schaltkreis mit BASIC Stamp verbinden

Die Versorgungs-leitungen des LED Schaltkreises werden mit den I/O Buchsen der BASIC Stamp verbunden.

Widerstände sind wichtig. Vergiss niemals Widerstände in einen Schaltkreis einzubauen. Damit wird verhindert, dass die Stromstärke zu stark ansteigt und ggf. Bauteile der Schaltung oder sogar das Board of Education oder HomeWork Board zerstört werden.

Programmgesteuerte LED

In unserem Beispielprogramm wird eine LED einmal pro Sekunde ein- und ausgeschaltet. Du wirst eine Reihe neuer Programmiertechniken kennenlernen und im weiteren Verlauf mit verschiedenen Teilen des Programms weiter arbeiten um seine Wirkungsweise besser verstehen zu können.

Programmbeispiel: LedOnOff.bs2

- √ Lade das Programm LedOnOff.bs2 in den BASIC Stamp Editor.
- √ Verbinde das Board of Education oder HomeWork Board mit der Energiequelle.
- √ Starte das Programm.
- √ Vergewisse dich, dass die LED einmal pro Sekunde ein- und ausschaltet.
- √ Schalte die Spannungsversorgung wieder ab, sobald die Arbeit beendet ist.

```
'What's a Microcontroller - LedOnOff.bs2
'Turn an LED on and off.  Repeat 1 time per second indefinitely.

'{$STAMP BS2}
```

```
'{$PBASIC 2.5}

DEBUG "The LED connected to Pin 14 is blinking!"

DO

  HIGH 14
  PAUSE 500
  LOW 14
  PAUSE 500

LOOP
```

Arbeitsweise von LedOnOff.bs2?

Der Befehl **`DEBUG "The LED connected to Pin 14 is blinking!"`** gibt den in Hochkomma angegebenen Text auf dem Debug Terminal aus. Mit dem Befehl **`HIGH 14`** wird der I/O Pin 14 der BASIC Stamp mit Vdd verbunden. Die LED beginnt zu leuchten.

`PAUSE 500` zwingt der BASIC Stamp eine Pause von 500 ms auf. In dieser Zeit passiert nichts; eine angeschaltete LED bleibt angeschaltet. Die Zahl 500 – das Argument des Befehls **`PAUSE`** besagt, dass die Pause 500/1000 Sekunden dauern wird. Im *BASIC Stamp Manual* wird das Argument des Befehls **`PAUSE`** auch als ***`Duration`*** (Dauer) bezeichnet.

Was versteht man unter einer Millisekunde? Eine Millisekunde ist 1/1000 einer Sekunde, abgekürzt mit ms. 1 Sekunde entsprechen 1000 ms.

Mit dem Befehl **`LOW 14`** verbindet die BASIC Stamp den I/O Pin P14 mit Vss – die LED wird abgeschaltet. **`LOW 14`** folgt der Befehl **`PAUSE 500`**, der die LED für eine halbe Sekunde abschaltet.

Die eben besprochenen Befehle liegen zwischen den beiden PBASIC Schlüsselwörtern **`DO`** und **`LOOP`**. Abbildung 2-12 zeigt, wie eine **`DO…LOOP`** Schleife arbeitet. Alle Befehle, die eine LED ein- bzw. ausschalten sind in einer solchen Schleife untergebracht und werden ständig wiederholt. Der Vorgang kann beenden werden, indem die Spannungsquelle vom *Board of Education* getrennt oder der Reset Schalter gedrückt wird. Bei einer Batterie kann man alternativ auch so lange warten, bis sie sich entladen hat.

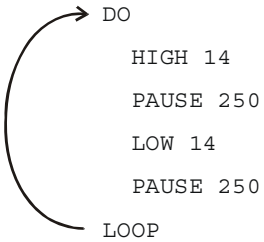


Abbildung 2-12
DO…LOOP Schleife

Die Befehle zwischen den Schlüsselwörtern DO und LOOP werden immer wieder ausgeführt.

Ein Schnittstellen-Diagnosetest

Einige Computer und Laptops können sich aus einer einmal gestarteten **DO...LOOP** Schleife eines PBASIC Programms nicht mehr befreien und „hängen sich auf". Diese Rechner verfügen über eine nicht standardisierte serielle Schnittstelle. Um überprüfen zu können, ob ein Rechner über eine standardisierte Schnittstelle verfügt, wird in das Programm LedOnOff.bs2 eine **DEBUG** Befehlszeile eingefügt; sie verhindert das Aufhängen des Rechners. In einem zweiten Schritt wird dann die Debug-Zeile gelöscht und das Verhalten des Rechners erneut geprüft.

- √ Öffne LedOnOff.bs2.
- √ Lösche die komplette **DEBUG** Zeile.
- √ Starte das veränderte Programm und beobachte die LED.

Blinkt die LED so wie im vorherigen Versuch, gibt es keine Schnittstellenprobleme mit dem Computer.

Geht die LED nur einmal an und wieder aus und hört das Blinken dann auf, verfügt dein Rechner über eine nicht-standardisierte serielle Schnittstelle. Löse in diesem Fall das serielle Kabel vom Board, drücke die Resettaste der BASIC Stamp und starte das Programm erneut.
In allen selbstgeschriebenen Programmen sollte die Befehlszeile:

```
DEBUG "Program Running!"
```

direkt nach den Compilerdirektiven einfügt werden. Damit wird das Debug Terminal und der COM Port geöffnet und gleichzeitig verhindert, dass sich das Programm nach einem Schleifenbefehl aufhängt.

Du bist dran – Blinkzeiten und Schleifen

Die An- und Abschaltzeiten einer LED lassen sich über das Argument ***Duration*** im Befehl **PAUSE** ändern. Wird das Argument ***Duration*** in beiden Befehlen auf 250 gesetzt, blinkt die LED zweimal pro Sekunde. Die **DO…LOOP** Schleife des Programms sieht nun wie folgt aus:

```
DO

  HIGH 14
  PAUSE 250
  LOW 14
  PAUSE 250

LOOP
```

√ Öffne LedOnOff.bs2

√ Setze ***Duration*** im Befehl **PAUSE** von 500 auf 250 und starte das Programm neu.

In einem weiteren Versuch soll versucht werden, die LED in drei Sekunden einmal ein- und auszuschalten. Dabei soll gelten: die Abschaltzeit ist doppelt so lang wie die Einschaltzeit. Um dies zu erreichen, musst nach einem **HIGH 14** ein **PAUSE** Befehl gesetzt werden, der den Controller 1 Sekunde (1000 ms) pausieren lässt. Entsprechend muss nach einem **LOW 14** Befehl der Controller für zwei Sekunden pausieren. Das folgende Programm zeigt die vollständige Schleife:

```
DO

  HIGH 14
  PAUSE 1000
  LOW 14
  PAUSE 2000

LOOP
```

√ Gib das Programm ein und führe es anschließend aus.

Versuche jetzt herauszufinden, wie kurz die Pausenzeiten sein dürfen, bei denen das Blinken der LED noch deutlich wahrgenommen wird. Blinkt die LED so schnell, dass nur noch ein permanentes Leuchten wahrgenommen wird, hängt das mit der Trägheit des

Auges zusammen. Im folgenden Test kannst du die Trägheitsschwelle deines Auges feststellen:

- √ Setze das Argument ***Duration*** des Befehls **PAUSE** auf 100.
- √ Starte das Programm neu und überprüfe, ob ein Flackern wahrzunehmen ist.
- √ Reduziere das Argument ***Duration*** um 5 und überprüfe aufs neue.
- √ Setze das Argument ***Duration*** so lange um jeweils 5 zurück bis die LED flackerfrei leuchtet.

Zum Schluss versuchen wir ein Programm zu schreiben, das einen LED Flash erzeugt, ein einmaliges Aufleuchten der LED. Damit lässt sich die Funktionalität einer **DO…LOOP** Schleife demonstrieren. Entferne dazu vorübergehend die Schleifenbefehle aus dem Programm, in dem du an den Zeilenanfang der Schlüsselwörter **DO** und **LOOP** ein Apostroph setzt (wir sagen dazu auch: Kommentiere die Zeilen aus.).

```
' DO

  HIGH 14
  PAUSE 1000
  LOW 14
  PAUSE 2000

' LOOP
```

- √ Änder das Programm so ab, wie oben gezeigt.
- √ Erkläre, wie das Programm arbeitet und warum die LED nur einmal aufleuchtet!

Kommentarzeilen einfügen: Ein Apostroph am Zeilenanfang macht aus einer Befehlszeile eine Kommentarzeile. Programmzeilen müssen nicht gelöscht und vielleicht später wieder zeitaufwendig eingegeben werden um das modifizierte Programm auszuprobieren.

ÜBUNG 3: ZÄHLEN UND WIEDERHOLEN

In der vorherigen Übung blinkte die LED entweder ununterbrochen oder sie leuchtete nur einmal auf und das Programm war beendet. Wie muss das Programm abgeändert werden, damit die LED zum Beispiel nur 10 mal an- und ausgeht? Dies lösen wir in der folgenden Übung. Die BASIC Stamp wird so programmiert, dass die LED nach zehn Wiederholungen automatisch zu blinken aufhört.

Bauteile Zählschaltung

Baue die Schaltung aus Abbildung 2-11, Seite 50, auf.

Anzahl der Wiederholungen

Es gibt mehrere Wege eine LED 10 mal ein- und auszuschalten; man benutzt dazu Schleifenbefehle. Wir kennen bereits die **`DO…LOOP`** Schleife, neu ist die sogenannte **`FOR…NEXT`** Schleife, die im Programm leichter darzustellen ist.
Die **`FOR…NEXT`** Schleife arbeitet mit einer Zählvariablen, die überprüft, wie häufig die Schleife bereits durchlaufen wurde. Der Name einer Zählvariablen wird vom Anwender selbst festgelegt. Im folgenden Beispielprogramm heißt die Zählvariable **`counter;`** mit ihr wird überprüft, wie häufig eine LED ein- und ausgeschaltet wurde.

Regeln für das Festlegen von Variablenamen:

1. Es dürfen keine von PBASIC benutzten Worte genommen werden. Dies sind sogenannte reservierte Worte, von denen du einige bereits kennst: **`DEBUG`**, **`PAUSE`**, **`HIGH`**, **`LOW`**, **`DO`**, und **`LOOP`**.
2. Ein Name darf keine Leerstelle enthalten.
3. Alle Namen müssen mit einem Buchstaben oder Zeichen beginnen; das erste Zeichen darf keine Zahl sein.
4. Eine Name darf nicht mehr als 33 Zeichen enthalten.

Programmbeispiel: LedOnOffTenTimes.bs2

In dem Programm LedOnOffTenTimes.bs2 wird gezeigt, wie eine LED mit einer **`FOR…NEXT`** Schleife 10 mal ein- und ausgeschaltet wird.

- √ Die Testschaltung aus Übung 2 sollte aufgebaut und startbereit sein.
- √ Gib das Programm LedOnOffTenTimes.bs2 in den BASIC Stamp Editor ein.
- √ Lege die Betriebsspannung an das Board of Education oder HomeWork Board.
- √ Führe das Programm aus.
- √ Überprüfe, ob die LED 10 mal ein und ausgeschaltet wird..
- √ Führe das Programm ein zweites mal aus und überprüfe auf dem Debug Terminal, ob über die Variable **`counter`** der Zählerstand der Schleife angezeigt wird. Hinweis: ein Programm kann sowohl über den Befehl *Run* als auch über den *Reset* Schalter des Board of Education oder HomeWork Board erneut gestartet werden.

```
' What's a Microcontroller - LedOnOffTenTimes.bs2
' Turn an LED on and off.  Repeat 10 times.

' {$STAMP BS2}
' {$PBASIC 2.5}

counter VAR Byte

FOR counter = 1 TO 10

  DEBUG ? counter

  HIGH 14
  PAUSE 500
  LOW 14
  PAUSE 500

NEXT

DEBUG "All done!"

END
```

Arbeitsweise von LedOnOffTenTimes.bs2

Mit der PBASIC Anweisung

```
counter VAR Byte
```

wird dem BASIC Stamp Editor mitgeteilt, dass das Programm das Wort **`counter`** als Variable des Typs Byte benutzt.

Was ist ein Byte? Ein Byte setzt sich aus 8 Bit zusammen. Ein Bit ist die kleinste Speichereinheit, die nur 0 oder 1 speichern kann. Ein Byte kann alle Zahlen zwischen 0 und 255 speichern. Die BASIC Stamp verfügt über vier verschiedene Variablentypen, die sich in ihren Wertebereichen unterscheiden:

Table 2-2: Variablentypen und ihr Wertebereich

Variablentyp	Wertebereich
Bit	0 und 1
Nib	0 bis 15
Byte	0 bis 255
Word	0 bis 65535

Das Formatierungszeichen ? vor einer Variablen in einem **DEBUG** Befehl teilt dem *Debug Terminal* mit, dass der Variablenname und der zugewiesene Wert angezeigt werden sollen. Mit der Befehlszeilte

```
DEBUG ? counter
```

wird sowohl der Name, gefolgt von einem Gleichheitszeichen, als auch der Wert der Variablen **counter** mit anschließendem Zeilenvorschub im *Debug Terminal* angezeigt.

Alle Befehlszeilen innerhalb der **FOR…NEXT** Schleife sind unten noch einmal dargestellt. In der ersten Anweisungszeile **FOR counter = 1 to 10** wird der BASIC Stamp mitgeteilt, dass sie die Zählvariable **counter** auf 1 setzen und anschließend alle Befehle ausführen soll, bis sie auf eine **NEXT** Anweisung trifft. Trifft die BASIC Stamp auf eine **NEXT** Anweisung, springt sie zu der Anweisungszeile mit der **FOR** Anweisung zurück. Gleichzeitig sorgt die **FOR** Anweisung dafür, dass der Wert der Zählvariablen um eins erhöht und anschließend überprüft wird, ob **counter** größer als 10 ist. Wenn das nicht der Fall ist, wird der oben besprochene Ablauf wiederholt. Hat mit der Erhöhung um eins **counter** den Wert 11 erreicht, springt das Programm in die erste Zeile, die der **NEXT** Anweisung folgt.

```
FOR counter = 1 to 10

  DEBUG ? counter

  HIGH 14
  PAUSE 500
  LOW 14
  PAUSE 500

NEXT
```

Der Befehl nach der Anweisung **NEXT** lautet:

```
DEBUG "All done!"
```

Diese Befehlszeile wurde nur eingefügt, um zu zeigen, dass das Programm, nachdem die **FOR…NEXT** Schleife 10 mal durchlaufen wurde, tatsächlich mit der ersten Programmzeile, die der **NEXT** Anweisung folgt, weiterarbeitet,.

Du bist dran – Weitere Zählschleifen

√ Ersetze die Anweisung

`FOR counter = 1 to 10` durch: `FOR counter = 1 to 20`

im Programm LedOnOffTenTimes.bs2 und starte es erneut. Welche Unterschiede beobachtest du und hast du das so erwartet?

√ Verändere die **`FOR`** Anweisung ein zweites mal so, wie in der folgenden Zeile angegeben:

```
FOR counter = 20 to 120 STEP 10
```

Wie häufig leuchtete die LED auf? Welche Werte wurden im Debug Terminal angezeigt?

ÜBUNG 4: AUFBAU UND TEST EINER ZWEITEN LED SCHALTUNG

Mit Hilfe von LED Leuchtanzeigen kann der Anwender eine Vielzahl an Informationen über das Verhalten eines Gerätes erfahren. Viele Geräte verfügen über zwei, drei oder noch mehr LEDs, die dem Nutzer anzeigen, ob die Maschine betriebsbereit ist oder nicht, ob eine Fehlfunktion aufgetreten ist, ob ein Arbeitsauftrag abgearbeitet wurde oder sich die Maschine noch in der Ausführungsphase befindet und vieles mehr.

In dieser Übung wird noch einmal auf den LED Schaltungstest aus Übung 1 mit einem zweiten LED Kreis Bezug genommen. Das Beispielprogramm aus Übung 2 muss anschließend angepasst werden, um sicherzustellen, dass auch die zweite LED Schaltung mit der BASIC Stamp richtig verbunden ist. In einem letzten Schritt wird das Programm aus Übung 2 so verändert, dass beide LEDs zusammen reagieren.

Zusätzliche Bauteile

Zusätzlich zu den Bauteilen aus Übung 1 und Übung 2 werden benötigt:

1 LED - gelb
1 Widerstand – 470 Ω (gelb-violett-braun)

Aufbau und Test einer zweiten LED Schaltung

In Übung 1 wurde der erste LED Schaltkreis noch "mit der Hand" ausgetestet, bevor er mit der BASIC Stamp verbunden wurde. Vor dem Einsatz einer zweiten LED Schaltung an der BASIC Stamp muss vorher wiederum ein Bauteiltest gemacht werden.

- √ Trenne die Energiequelle vom Board of Education oder dem HomeWork Board.
- √ Baue die zweite LED Schaltung, wie in Abbildung 2-13 gezeigt, auf.
- √ Verbinde die Energiequelle mit dem Board of Education oder HomeWork Board.
- √ Leuchtet die LED in der neuen Schaltung auf? Wenn ja, setze die Arbeit fort. Wenn nicht, werden in Übung 1 möglichen Fehlerquellen und deren Beseitigung aufgezeigt.

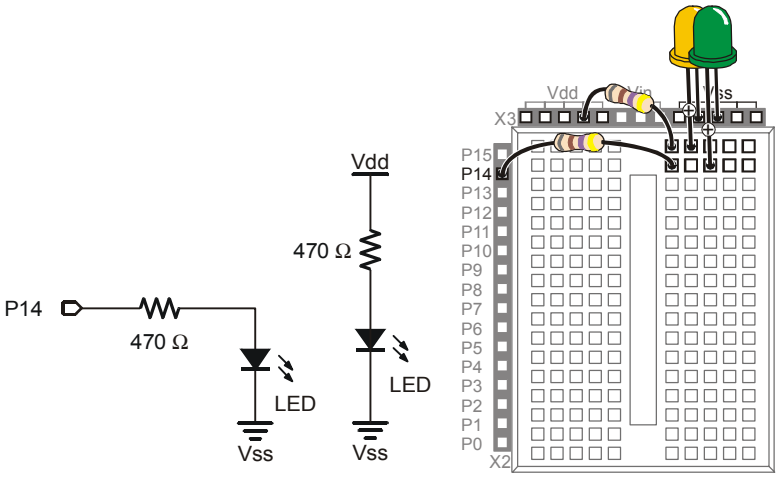


Abbildung 2-13 Manuelle Testschaltung für die zweite LED

- √ Trenne die Energiequelle vom Board of Education oder HomeWork Board.
- √ Veränder die zweite LED Schaltung so, dass das Anschlussbeinchen des Widerstandes jetzt mit (Input) Buchse P15 verbunden wird, wie in Abbildung 2-14 gezeigt.

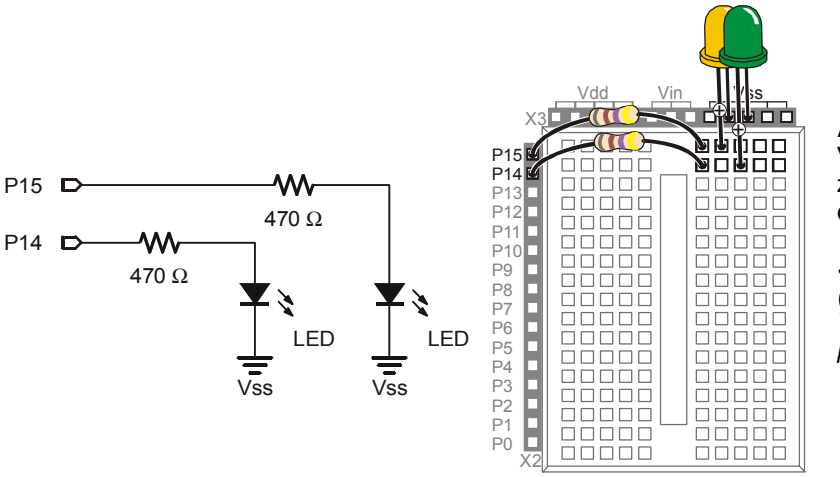


Abbildung 2-14
Verbinde die zweite LED mit der BASIC Stamp

Schaltskizze (links) und Verdrahtungsplan (rechts).

LED Schaltungstest mit einem Programm

In Übung 2 haben wir mit Hilfe eines Programms und der Befehle **`HIGH`** und **`LOW`** die mit P14 verbundene LED Schaltung gesteuert. Die Befehle müssen jetzt so angepasst werden, dass sie auch mit einer Schaltung an P15 zusammenarbeiten. Dies geschieht mit Hilfe der Aufrufe **`HIGH 15`** und **`LOW 15`**.

Programmbeispiel: TestSecondLed.bs2

- √ Gib das Programm TestSecondLed.bs2 in den BASIC Stamp Editor ein.
- √ Verbinde das Board of Education oder HomeWork Board mit einer Energiequelle.
- √ Führe Run TestSecondLED.bs2 aus.
- √ Überprüfe, ob die LED Schaltung an P15 blinkt. Ist das der Fall, beschäftige dich mit der nächsten Übung (Zwei LEDs ansteuern). Funktioniert die neue LED Schaltung an P15 nicht (LED blinkt nicht), überprüfe zunächst die Verdrahtung und anschließend das Programm auf Fehler. Korrigiere sie und starte das Programm erneut.

```
' What's a Microcontroller - TestSecondLed.bs2
' Turn LED connected to P15 on and off.
' Repeat 1 time per second indefinitely.

' {$STAMP BS2}
' {$PBASIC 2.5}
```

```
DEBUG "Program Running!"

DO

  HIGH 15
  PAUSE 500
  LOW 15
  PAUSE 500

LOOP
```

Zwei LEDs ansteuern

Es ist auch möglich zwei LEDs gleichzeitig zum Blinken zu bringen. Dies geschieht über zwei aufeinanderfolgende **HIGH** Anweisungen vor dem ersten **PAUSE** Befehl. Mit dem ersten **HIGH** Befehl wird P14 auf 1 und mit dem folgenden P15 auf 1 gesetzt. Für das Abschalten der LEDs werden ebenso zwei **LOW** Anweisungen benötigt. Du hast bereits richtig erkannt, dass mit dieser Vorgehensweise die beiden LEDs nicht gleichzeitig sondern hintereinander ein- bzw. ausgeschaltet werden. Dies geschieht aber so schnell – mehr als eine Millisekunde wird für das aufeinander folgende Schalten nicht benötigt – dass das träge menschliche Auge davon gar nichts mit bekommt.

Programmbeispiel: FlashBothLeds.bs2

- √ Gib das Programm FlashBothLeds.bs2 in den BASIC Stamp Editor ein.
- √ Führe das Programm aus.
- √ Überzeuge dich, dass beide LEDs gleichzeitig ein- bzw. ausgeschaltet werden.

```
' What's a Microcontroller - FlashBothLeds.bs2
' Turn LEDs connected to P14 and P15 on and off.

' {$STAMP BS2}
' {$PBASIC 2.5}

DEBUG "Program Running!"

DO

  HIGH 14
  HIGH 15
  PAUSE 500
  LOW 14
  LOW 15
  PAUSE 500
```

```
LOOP
```

Du bist dran – LED Blinker

Mit Hilfe der beiden Befehle **HIGH** und **LOW,** mit denen die I/O Pins angesteuert werden, lassen sich beide LEDs auch nacheinander (eine LED an – die zweite aus und umgekehrt) ein- bzw. ausschalten.

√ Veränder das Programm FlashBothLeds.bs2 in der **DO ... LOOP** Schleife so, wie unten angegeben:

```
HIGH 14
LOW 15
PAUSE 500
LOW 14
HIGH 15
PAUSE 500
```

√ Führe das veränderte Programm FlashBothLeds.bs2 aus und überprüfe, ob die LEDs nacheinander aufleuchten.

ÜBUNG 5: ANSTEUERUNG EINER DUO-LED

Das in Abbildung 2-15 dargestellte Gerät ist ein Sicherheitsmonitor für elektronische Schließsysteme. Sobald der richtige Zugangscode eingegeben wird, verändert sich die angezeigt Farbe der LED und die Tür entriegelt. Eine LED, die aus zwei LEDs unterschiedlicher Farbe aufgebaut ist, heißt DUO-LED. In dieser Übung werden wir zwei Fragen beantworten:

1. Wie kommt es, dass die LED die Farbe wechselt?
2. Wie lässt sich eine solche LED mit der BASIC Stamp betreiben?

Abbildung 2-15
DUO LED in einem Sicherheitsgerät

Bei verschlossener Tür leuchtet die DUO LED rot. Wird die Tür über einen Zugangscode entriegelt, springt die Farbe der LED auf grün um.

Einführung DUO LED

Das Schaltsymbol einer Duo-LED und die Bauteilzeichnung zeigt Abbildung 2-16.

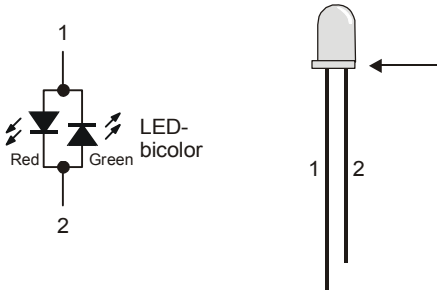


Abbildung 2-16
DUO LED

Schaltsymbol (links) und Bauteilzeichnung (rechts).

Wie dem Schaltsymbol einer Duo LED zu entnehmen ist, besteht sie aus zwei Dioden. Abbildung 2-17 zeigt, wie die Spannungsversorgung anzulegen ist, damit die LED rot aufleuchtet. Vertauscht man die Polung, leuchtet sie grün auf. Wie bei den anderen LEDs gilt auch hier, werden beide Anschlüsse eines Schaltkreises mit Vss verbunden, leuchtet die LED nicht.

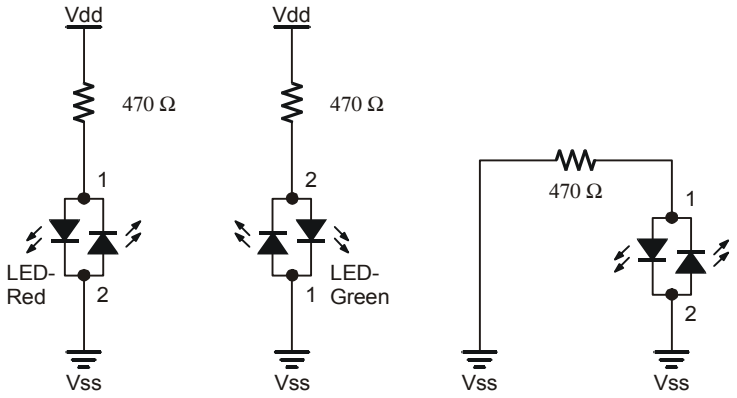


Abbildung 2-17
DUO LED mit Spannungsversorgung

Die Duo LED leuchtet rot (links), grün (Mitte) und gar nicht (rechts)

Bauteile DUO-LED Schaltung

1 Duo-LED
1 Widerstand 470 Ω (gelb-violett-braun)
1 Drahtbrücke

Aufbau und Test einer DUO LED Schaltung

Abbildung 2-18 zeigt den einfachen Testaufbau mit einer DUO LED.

- √ Trenne die Energiequelle vom Board of Education oder HomeWork Board.
- √ Baue die Schaltung wie links in Abbildung 2-18 gezeigt auf.
- √ Schließe die Energiequelle wieder an und überzeuge dich, dass die DUO LED rot aufleuchtet.
- √ Trenne die Energiequelle erneut vom Board.
- √ Änder die Schaltung so ab, dass sie dem Aufbau auf der rechten Seite in Abbildung 2-18 entspricht.
- √ Schließe die Spannungsquelle erneut an.
- √ Überzeuge dich, dass die DUO LED jetzt grün aufleuchtet.
- √ Trenne die Energiequelle vom Board.

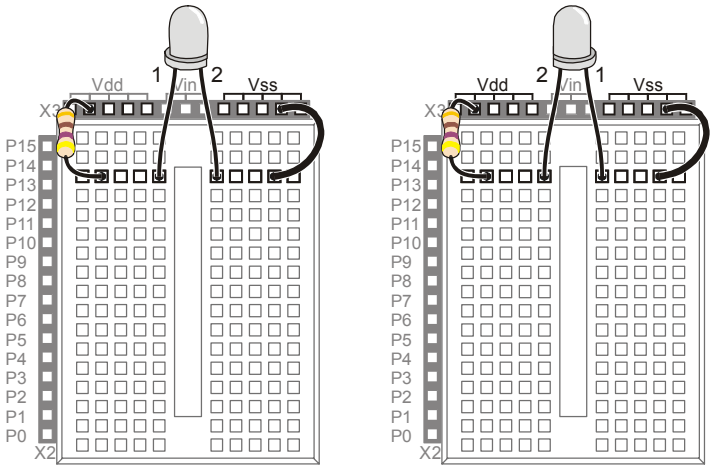


Abbildung 2-18
Einfacher DUO LED Test

DUO LED leuchtet rot (links) und grün (rechts).

Für den Anschluss einer DUO LED an die BASIC Stamp werden zwei I/O Anschlüsse benötigt. Nachdem die DUO LED mit dem einfachen Versuchsaufbau getestet wurde, verbinde die Schaltung jetzt mit der BASIC Stamp so, wie in Abbildung 2-19 gezeigt.

√ Verbinde die DUO LED Schaltung mit der BASIC Stamp wie in Abbildung 2-19 gezeigt.

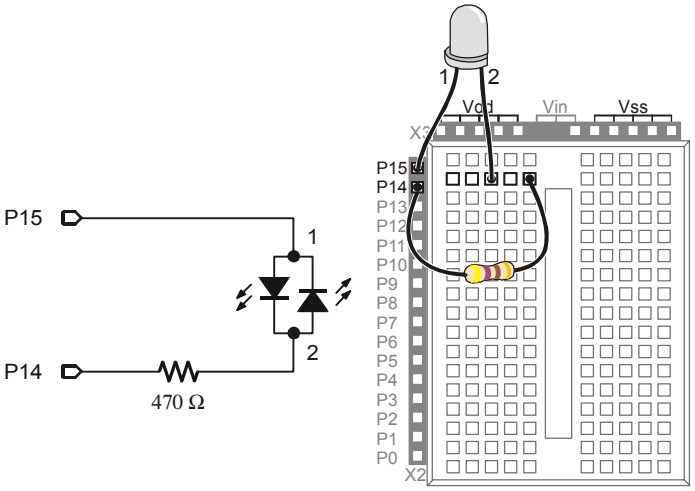


Abbildung 2-19
DUO LED Schaltung wird mit BASIC Stamp verbunden.

Ansteuerung einer DUO-LED mit der BASIC Stamp

Abbildung 2-20 zeigt, wie mit den Anschlüssen P15 und P14 der Stromfluss durch eine DUO LED gesteuert werden kann. In der oberen Schaltskizze erkennt man, dass der Strom durch die rote LED fließt, wenn P15 mit Vdd (hohes Potential) und P14 mit Vss (niedriges Potential) verbunden wird. Als Eselsbrücke kannst du dir merken: Der Stromfluss und das Diodensymbol zeigen bei einer durchgeschalteten Diode immer in die gleiche Richtung. Die grüne LED ist bei dieser Beschaltung in Sperrichtung gepolt, sie leuchtet nicht. Die DUO LED leuchtet rot.

Wird P15 mit Vss (niedriges Potential) und P14 mit Vdd (hohes Potential) verbunden, so wie es die untere Schaltskizze wiedergibt, dann ist die grüne Diode in Durchlassrichtung und die rote in Sperrrichtung geschaltet. Die rote LED erlischt und die grüne leuchtet auf.

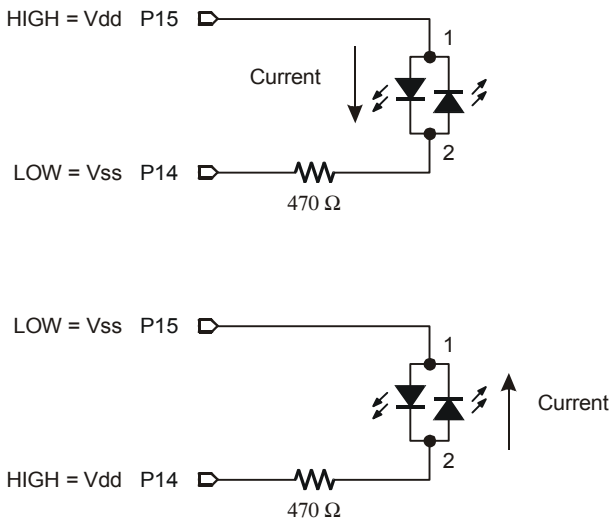


Abbildung 2-20
Einfacher DUO LED Test

Stromfluss durch die rote LED (oberes Bild) und die grüne LED (unteres Bild).

Abbildung 2-20 liefert alle Hinweise, die es uns ermöglichen, die BASIC Stamp so zu programmieren, dass die DUO LED in zwei verschiedenen Farben aufleuchtet. Die obere Schaltskizze zeigt, warum mit den Befehlen **`HIGH 15`** und **`LOW 14`** die DUO LED rot und in der unteren Schaltskizze mit **`LOW 15`** und **`HIGH 14`** grün aufleuchtet. Abgeschaltet wird die LED wenn beide Anschlüsse (P14 und P15) auf niedriges Potential gelegt werden: **`LOW 15`** und **`LOW 14`**.

Eine DUO LED schaltet auch dann ab, wenn sowohl P14 als auch P15 auf HIGH gesetzt werden. Warum? Da der Ladungsdruck an beiden Polen gleich groß ist, unabhängig ob die beiden I/O Pins auf HIGH oder LOW liegen, findet eine Ladungsbewegung nicht statt.

Programmbeispiel: TestBiColorLED.bs2

- √ Verbinde die Schaltung mit der Energiequelle.
- √ Gib das Programm TestBiColorLed.bs2 in den BASIC Stamp Editor ein.
- √ Führe das Programm aus.
- √ Überprüfe, ob die LED den folgenden Zyklus durchläuft: rot, grün, LED schaltet ab.

```
' What's a Microcontroller - TestBiColorLed.bs2
' Turn bi-color LED red, then green, then off in a loop.

' {$STAMP BS2}
' {$PBASIC 2.5}

DEBUG "Program Running!"

DO

  HIGH 15           ' Red
  LOW 14
  PAUSE 500

  LOW 15            ' Green
  HIGH 14
  PAUSE 500

  LOW 15            ' Off
  LOW 14
  PAUSE 500

LOOP
```

Du bist dran – Lichtspiele

In Übung 3 haben wir eine Variable **`counter`** kennengelernt. Mit ihr legten wir fest, wie häufig eine LED aufleuchten soll. Was passiert nun, wenn mit dem Wert von **`counter`** das Argument ***`Duration`*** im Befehl **`PAUSE`** angesteuert und damit Einfluss auf die Pausendauer genommen wird?

√ Benenne die Datei TestBiColorLed.bs2 in TestBiColorLedYourTurn.bs2 um und speicher sie sicherheitshalber ab.

√ Deklariere vor der **DO** Anweisung im Programm eine Zählvariable:

```
counter VAR BYTE
```

√ Füge in die **DO…LOOP** Schleife eine **FOR…NEXT** Schleife, wie unten dargestellt, ein.

```
FOR counter = 1 to 50

  HIGH 15
  LOW 14
  PAUSE counter

  LOW 15
  HIGH 14
  PAUSE counter

NEXT
```

Die Programmzeilen sollten jetzt wie folgt aussehen:

```
counter VAR BYTE

DO

  FOR counter = 1 to 50

    HIGH 15
    LOW 14
    PAUSE counter

    LOW 15
    HIGH 14
    PAUSE counter

  NEXT

LOOP
```

Zu Beginn des Schleifendurchgangs der **FOR…NEXT** Schleife beträgt der **PAUSE** Wert (***Duration*** Argument) nur eine Millisekunde. Jeder weitere Schleifendurchgang erhöht

die Pausendauer um eine Millisekunde bis schließlich 50 Millisekunden erreicht sind. Innerhalb der **DO…LOOP** Schleife wird die **FOR…NEXT** Schleife immer wieder neu durchlaufen.

√ Führe das abgeänderte Programm aus und beobachte den Effekt.

ZUSAMMENFASSUNG

Die BASIC Stamp lässt sich so programmieren, dass eine Schaltung mit einer LED sich ein- und ausschalten lässt. Leuchtdiodenanzeigen sind in vielen Geräten, wie Monitoren, CD-ROM Laufwerken usw. zu finden. Wir haben gelernt, wie Anode und Kathode einer LED bestimmt werden können und das der Halbleiter stets mit einem strombegrenzenden Widerstand aufgebaut werden muss. Ebenso sind wir in der Lage über eine Kodierungstabelle den Wert eines Widerstandes an Hand der aufgedruckten Farbringe zu bestimmen.

Die BASIC Stamp schaltet über einen I/O Pin eine LED in einer Schaltung ein oder aus. Dabei wird der Pin wahlweise an Vdd oder Vss gelegt. Mit dem **HIGH** Befehl werden die I/O Pins einer BASIC Stamp intern mit Vdd und mit dem **LOW** Befehl mit Vss verbunden. Der Befehl **PAUSE** wird dann eingesetzt, wenn die BASIC Stamp über einen festen Zeitraum keine weiteren Befehle ausführen soll. Mit diesen Methoden konnten wir unsere LEDs für einen bestimmten Zeitraum ein- bzw. ausschalten. Die Zeitdauer haben wir mit Hilfe einer Zahlenangabe im Argument ***Duration*** des **PAUSE** Befehls beeinflusst.

Mit **DO…LOOP** wird eine Endlosschleife aufgebaut. Alle Befehle zwischen den Schlüsselwörtern **DO** und **LOOP** werden ständig wiederholt. Obwohl es sich hier um eine Endlosschleife handelt, kann das Programm jederzeit neu gestartet werden, wenn die Spannungsversorgung unterbrochen und anschließend wieder hergestellt wird oder indem man die Resettaste drückt. Wird ein neues Programm in die BASIC Stamp geladen, überschreibt der Ladevorgang das alte Programm und löscht es.

Stromrichtung und Polarität der Spannungsversorgung wurden im Zusammenhang mit einer DUO LED angesprochen. Wird eine LED Schaltung an eine Spannungsquelle angeschlossen, dann fließt der Strom nur in einer Richtung und eine DUO LED leuchtet in einer bestimmten Farbe. Werden die Anschlusspole dann vertauscht, kehrt sich die Stromrichtung im Schaltkreis um und die LED leuchtet in einer anderen Farbe.

Verständnisfragen

1. Wie heißt der griechische Buchstabe: Ω? Für welche physikalische Größe wird er als Bezeichner der Einheit benutzt?
2. Bei welchem Widerstandswert fließt ein größerer Strom durch die Schaltung: 470 Ω oder 1000 Ω?

3. Wie werden auf dem Schaltungsbrett zwei Drähte miteinander verbunden? Lassen sich auf dem Schaltungsbrett auch vier Drähte miteinander verbinden?
4. Was muss beachtet werden, bevor Änderungen an Schaltungsaufbauten auf dem Steckbrett vorgenommen werden?
5. Wie groß ist die Pause bei `PAUSE 10000`?
6. Mit welchem Befehl wird eine BASIC Stamp für eine ganze Minute angehalten?
7. Welche unterschiedlichen Variablentypen gibt es?
8. Kann die Zahl 500 mit einer Variablen vom Typ Byte dargestellt werden?
9. Was bewirkt der Befehl `HIGH 7` ?

Aufgaben

1. Zeichne die Schaltskizze des LED Schaltkreises aus Übung 2, dabei soll die Schaltung mit P13 statt mit P14 verbunden sein. Erkläre, wie das Programm **Programmbeispiel:** LedOnOff.bs2 auf Seite 50 abgeändert werden muss, damit die LED viermal pro Sekunde ein- und ausgeschaltet wird.
2. Erkläre, wie das Programm LedOnOffTenTimes.bs2 abgeändert werden muss, damit die LED 5000 mal ein- und ausgeschaltet wird. Hinweis: es müssen nur zwei Programmzeilen verändert werden.

Projekt

1. Stelle mit einer gelben LED und einer DUO LED eine Schaltung her, die einen Countdown von 10 Sekunden ausführt. Die DUO LED soll die ersten drei Sekunden rot aufleuchten. Danach schaltet sie auf grün um. Sobald die DUO LED auf grün umschaltet, geht die gelbe LED über einen Zeitraum von 10 Sekunden einmal pro Sekunde an und aus (Countdown läuft). Am Ende des Countdown – nach 10 Sekunden – schaltet die DUO LED auf rot zurück.

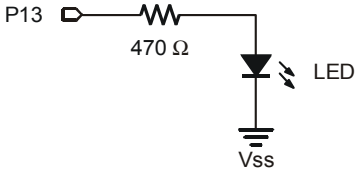
Lösungen

Q1. Omega steht für die Einheit Ohm, einer Einheit für den elektrischen Widerstand.
Q2. 470 Ω Widerstand: je höher der Widerstandswert, desto geringer der Stromfluss durch die Schaltung.
Q3. Zum Verbinden zweier Drähte stecke diese nebeneinander in eine 5-er Buchse. Es lassen sich insgesamt bis zu 5 Drähte miteinander verbinden.
Q4. Die Energiequelle muss abgetrennt werden.
Q5. 10 Sekunden.
Q6. PAUSE 60000
Q7. Bit, Nib, Byte und Word

Q8. Nein. Der größte darstellbare Zahlenwert im Format Byte ist die Zahl 255. Die Zahl 500 benötigt im 2-er System (binäres System) insgesamt 9 Bit.

Q9. **`HIGH 7`** veranlasst die BASIC Stamp intern den I/O Pin P7 mit Vdd zu verbinden.

E1. Der Befehl **`PAUSE Duration`** muss auf 500ms / 4 = 125ms reduziert werden. Um den I/O Pin P13 zu nutzen, mussen **`HIGH 14`** und **`LOW 14`** durch **`HIGH 13`** and **`LOW 13`** ersetzt werden.

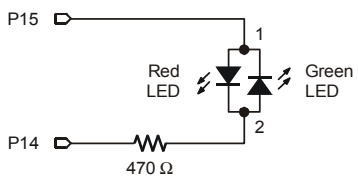


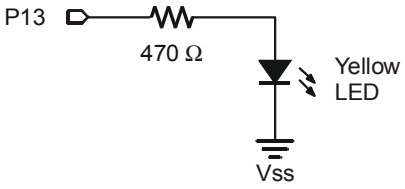

```
DO
  HIGH 13
  PAUSE 125
  LOW 13
  PAUSE 125
LOOP
```

E2. Der Variablen **`counter`** muss der Typ **`Word`** zugeordnet werden, die **`FOR`** Anweisung muss so verändert werden, dass sie von 1 bis 5000 zählt.

```
counter VAR Word
FOR counter = 1 to 5000
  DEBUG ? counter, CR
  HIGH 14
  PAUSE 500
  LOW 14
  PAUSE 500
NEXT
```

P1. Die DUO LED Schaltung links ist unverändert aus Abbildung 2-20 auf Seite 66 übernommen worden. Die Schaltung mit der gelben LED basiert auf Abbildung 2-11 von Seite 50. P14 wurde in P13 geändert.






```
' What's a Microcontroller - Ch02Prj01_Countdown.bs2
' 10 Second Countdown with Red, Yellow, Green LED
' Red/Green:  Bicolor LED on P15, P14. Yellow: P13
' {$STAMP BS2}
```

```
' {$PBASIC 2.5}
DEBUG "Program Running!"

counter VAR Byte

' Red for three seconds                 ' Bi-color LED Red
  HIGH 15
  LOW 14
  PAUSE 3000

' Green for 10 seconds...
  LOW 15                                ' Bi-color LED Green
  HIGH 14

' ...while the yellow LED is flashing
  FOR counter = 1 TO 10
    HIGH 13                             ' Yellow LED on
    PAUSE 500
    LOW 13                              ' Yellow LED off
    PAUSE 500
  NEXT

' Red stays on                          ' Bi Color LED Red
  HIGH 15
  LOW 14
```

Weiterführende Bücher

Die folgenden Bücher können kostenlos von der Parallax Website geladen werden und sind auch auf der Parallax CD zu finden.

***"BASIC Stamp Manual"*, User Manual, Version 2.0c, Parallax Inc., 2000**

In dem *BASIC Stamp Manual* befinden sich weitere Beispiele zum ausprobieren und zusätzliche Informationen, die die Befehle: **`HIGH`**, **`LOW`**, **`PAUSE`**, die Formatierungsanweisung **`DEBUG ?`** und **`FOR…NEXT`** näher erläutern.

***"Basic Analog and Digital"*, Student Guide, Version 2.0, Parallax Inc., 2003**

Basic Analog and Digital arbeitet mit LEDs und beschreibt binäre Zähler, analoge Abläufe und weist neue Wege auf, die Helligkeit von LEDs anzupassen.

“*BASIC Stamp Editor Help File*”, PBASIC 2.5 Version 2.0 Parallax Inc., 2003

Die Hilfedatei von PBASIC 2.5 enthält Hinweise zur `DO…LOOP` Schleife, die neu in PBASIC 2.5 aufgenommen wurde und nicht im BASIC Stamp Manual enthalten ist. Klicke auf das Buchsymbol in der Task Bar deines BASIC Stamp Editor und wähle PBASIC Reference aus dem Menü im linken Fenster. Damit wird die alphabetisch sortierte PBASIC Befehlsreferenzliste geoffnet. Hier erhälst du umfassende Informationen zu jedem Befehl.

Kapitel 3: Digitale Eingabe mit Taster

TASCHENRECHNER, GAME-BOYS UND ANDERE GERÄTE

Überlege einmal, wie viele Geräte mit Tastern du täglich benutzt. Einige Beispiele sollen helfen: Computer, Maus, Taschenrechner, Mikrowelle, Fernbedienung des Fernsehers, Mobiltelefon, Gameboy, MP3-Player etc.. In jedem dieser Geräte befindet sich ein Mikrocontroller, der die Taster überwacht und reagiert, sobald einer betätigt wird (d.h. ein Schaltkontakt geschlossen bzw. geöffnet wird). Wenn du dieses Kapitel durchgearbeitet haben wirst, hast du ausreichend Erfahrung gesammelt, wie man Schaltungen mit Tastern aufbaut und wie man die BASIC Stamp so programmiert, dass sie die Taster überwacht und auf Tastendruck entsprechend reagiert.

SENDEN UND EMPFANGEN VON HIGH/LOW SIGNALEN

In Kapitel2 wurde die BASIC Stamp bereits so programmiert, dass sie HIGH und LOW Signale aussendete. Über eine LED Schaltung wurden diese Signalzustände ausgegeben. HIGH/LOW Signale aussenden heißt, das ein I/O Pin der BASIC Stamp als Ausgabekanal genutzt wurde. In diesem Kapitel werden wir einen BASIC Stamp I/O Pin als Eingabekanal nutzen. Eingabekanäle warten auf HIGH/LOW Signale, während Ausgabekanäle HIGH/LOW Signale senden. Mit einer Taster-Schaltung werden wir an die BASIC Stamp solche Signale senden; der Controller muss von uns nur noch so programmiert werden, dass er erkennt, ob ein Taster gedrückt worden ist oder nicht.

Weitere Darstellungen für SENDEN, HIGH/LOW und EMPFANGEN: Das Senden von high/low Signalen kann man sich auf verschiedene Art und Weise zu erklären versuchen. Signale senden hängt eng zusammen mit Begriffen wie Übertragen, Steuern oder Schalten. Statt von HIGH/LOW zu sprechen, benutzen wir häufig auch die Darstellung 0 und 1, AN und AUS und verbinden damit binäre, TTL, CMOS oder booleansche Signale.

ÜBUNG 1: TASTERVERHALTEN MIT EINER LED SCHALTUNG PRÜFEN

Wenn sich mit einem Schalter HIGH/LOW Signale an eine BASIC Stamp übertragen lassen, lässt sich dann auch eine LED mit einem Taster über die BASIC Stamp steuern? Die Antwort ist ein klares Ja. Wir werden das in der nächsten Übung zeigen.

Einführung Taster

Abbildung 3-1 zeigt das Schaltsymbol und die Bauteilzeichnung eines normalen offenen Tasters. Jeder der beiden Schaltkontakte verfügt über zwei Anschlüsse, d.h. dass z.B. die

Anschlusspins 1 und 4 sowie 2 und 3 jeweils an einem Kontakt liegen. Der Grund, warum jeder Taster über vier Anschlussdrähte verfügt, liegt in der Stabilität des Schalters begründet. Mit vier Auflagepunkten hält er dem Fingerdruck auf den Taster besser stand, als wenn er nur über zwei verfügen würde.

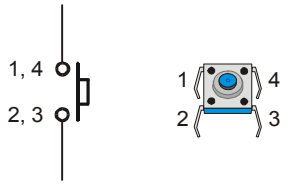


Abbildung 3-1
Taster als einfacher Schließer

Schaltzeichen (links) und Bauteilzeichnung (rechts)

Auf der linken Seite von Abbildung 3-2 ist ein einfacher Schließer dargestellt. Im nicht betätigten Zustand sind die Anschlusspunkte 1,4 und 2,3 nicht verbunden; ein Schließer in einem Schaltkreis unterbricht den Stromkreis – der Stromkreis ist offen. Wird der Taster gedrückt, werden die Anschlüsse 1,4 und 2,3 miteinander verbunden. Der Taster (Schließer) ist geschlossen und ein Strom kann im Stromkreis fließen.

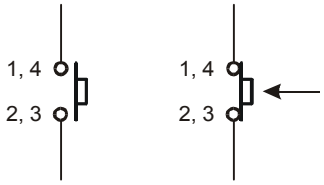


Abbildung 3-2
Taster als einfacher Schließer

unbetätigt (links) und betätigt (rechts)

Bauteile für Tasterschaltung

1 LED – beliebige Farbe
1 Widerstand - 470 Ω (gelb-violett-braun)
1 Taster als Schließer
1 Drahtbrücke

Testschaltung mit Taster

Abbildung 3-3 zeigt eine Schaltung, mit der die Arbeitsweise eines Taster ausprobiert wird.

Unterbrich jedesmal die Spannungsversorgung zum Board of Education oder BASIC Stamp HomeWork Board bevor irgendwelche Änderungen an der Testschaltung vorgenommen werden. Von jetzt ab werden wir diesen Hinweis nicht mehr geben.

Unterbrich die Spannungsversorgung zum Board of Education oder BASIC Stamp HomeWork Board jedes Mal, wenn ein neues Programm in die BASIC Stamp geladen wird.

√ Baue die Schaltung nach Abbildung 3-3 auf.

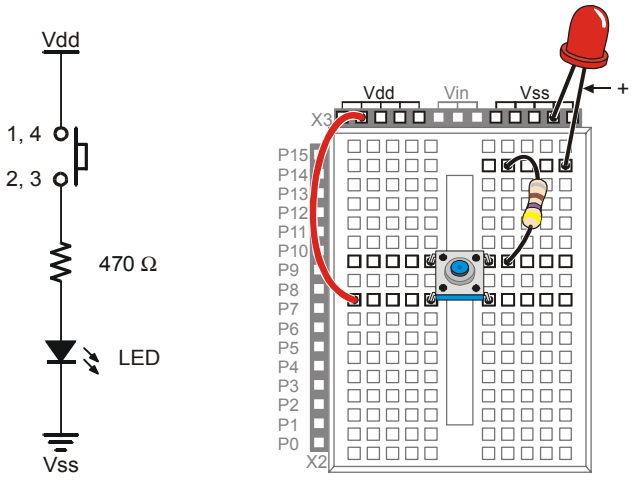


Abbildung 3-3
Testschaltung mit Taster

Den Taster prüfen

Wird der Taster nicht betätigt, bleibt die LED dunkel. Bei korrekter Verdrahtung leuchtet die LED auf, sobald der Taster gedrückt wird.

Warnzeichen: Wenn die Power LED des Board of Education flackert, schwächer leuchtet oder ausgeht, sobald die Spannungsversorgung anliegt, liegt ein Kurzschluss vor. Sollte das der Fall sein, trenne sofort die Spannungsversorgung von der Schaltung und versuche den Fehler in der Schaltung zu finden.

Die Power LED des HomeWork Board arbeitet anders. Sie leuchtet nur solange ein Programm ausgeführt wird. Sobald ein Programm beendet ist, wird die Power LED ausgeschaltet.

√ Überprüfe, ob die LED der Testschaltung ausgeschaltet ist.

√ Drücke auf den Taster und überzeuge dich, dass die LED leuchtet, so lange der Taster gedrückt bleibt.

Arbeitsweise der Tasterschaltung

Auf der linken Seite in Abbildung 3-4 wird dargestellt, was in der Schaltung passiert, wenn der Taster nicht betätigt wird. Die LED ist nicht mit Vdd verbunden, es handelt sich um einen nicht geschlossenen Stromkreis. Wird der Taster gedrückt, schließt sich der Schalter, die Elektronen fließen vom Plus- (Vdd) zum Minuspol (Vss) und die LED beginnt zu leuchten.

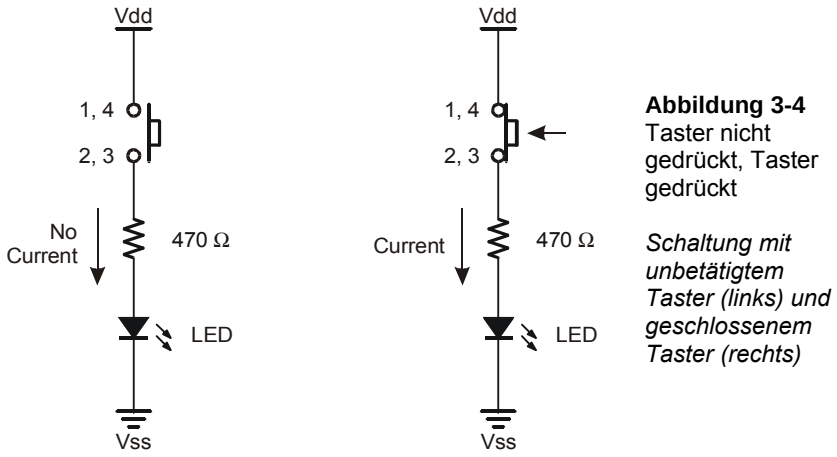


Abbildung 3-4
Taster nicht gedrückt, Taster gedrückt

Schaltung mit unbetätigtem Taster (links) und geschlossenem Taster (rechts)

Du bist dran – Kurzschlussschaltung

Abbildung 3-5 zeigt eine weitere Schaltung in der die LED sich etwas anders verhält. Wird der Taster nicht betätigt, leuchtet die LED, bei gedrücktem Taster erlischt sie. Die Erklärung, warum sie erlischt, sobald der Taster gedrückt wird, erfolgt über eine Betrachtung des Stromflusses. Ein Strom nimmt immer den Weg des geringsten Widerstandes. Wird der Taster gedrückt, ist der Widerstand zwischen den Anschlüssen 1,4 und 2,3 verschwindend gering, so dass der komplette Strom durch den Taster statt durch die LED fließt (Kurzschlussschaltung).

√ Baue die Schaltung nach Abbildung 3-5 auf.
√ Führe mit dieser Schaltung die gleichen Tests durch wie mit der vorherigen Tasterschaltung.

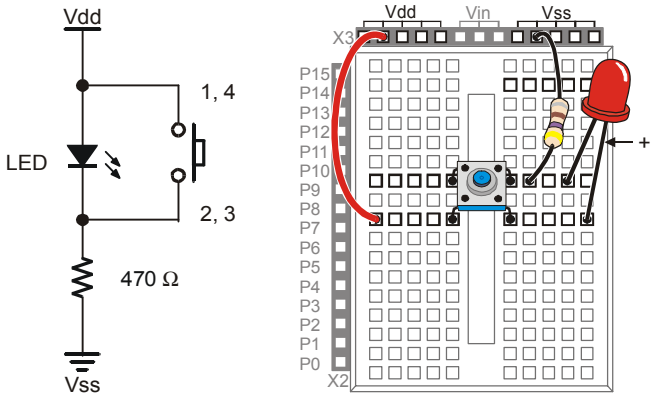


Abbildung 3-5
LED wird über Taster kurzgeschlossen

Verträgt eine LED das? Bis jetzt haben wir die Kathode einer LED immer mit Vss verbunden. In dieser Schaltung befindet sich die LED an einer anderen Stelle inerhalb der Schaltung, wobei die Anode mit Vdd verbunden ist. Zwischen Vdd und Vss liegen 5 Volt an. Die Leuchtdiode benötigt für den Betrieb immer 1,6 Volt, über dem Vorschaltwiderstand müssen 3,4 Volt abfallen.

ÜBUNG 2: TASTERABFRAGE MIT DER BASIC STAMP

In dieser Übung wird die Schaltung mit Taster mit der BASIC Stamp verbunden und auf einem Display der Zustand des Taster (gedrückt/nicht gedrückt) angezeigt. Dazu muss ein PBASIC Programm geschrieben werden, das den Zustand des Tasters abfragt und anschließend auf dem *Debug Terminal* ausgibt.

Bauteile der Tasterschaltung

1 Taster als Schließer
1 Widerstand – 220 Ω (rot-rot-braun)
1 Widerstand – 10 kΩ (braun-schwarz-orange)
2 Verbindungsdrähte

Tasterschaltung mit BASIC Stamp verbinden

Abbildung 3-6 zeigt eine Schaltung mit Taster, die mit dem I/O Pin P3 einer BASIC Stamp verbunden ist.

√ Baue die Schaltung aus Abbildung 3-6 auf.

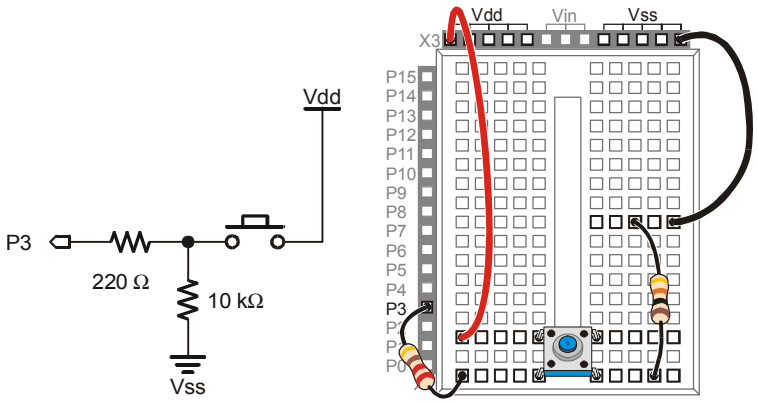


Abbildung 3-6
Schaltung mit Taster, der mit I/O Pin P3 verbunden ist

Im Verdrahtungsplan verbindet der 220 Ω Widerstand auf der linken Seite den Taster mit P3 während der 10 kΩ Widerstand rechts den Taster mit Vss verbindet.

In Abbildung 3-7 wird dargestellt, was die BASIC Stamp "sieht", wenn der Taster gedrückt wird (oberes Bild) und wenn er unbetätigt bleibt (unteres Bild). Wird der Taster gedrückt, erkennt die BASIC Stamp, dass Vdd mit P3 verbunden ist. Am I/O Pin P3 liegt jetzt eine 1 (HIGH) an. Bleibt der Taster unberührt, erkennt die BASIC Stamp das am I/O Pin P3 eine 0 (LOW) anliegt.

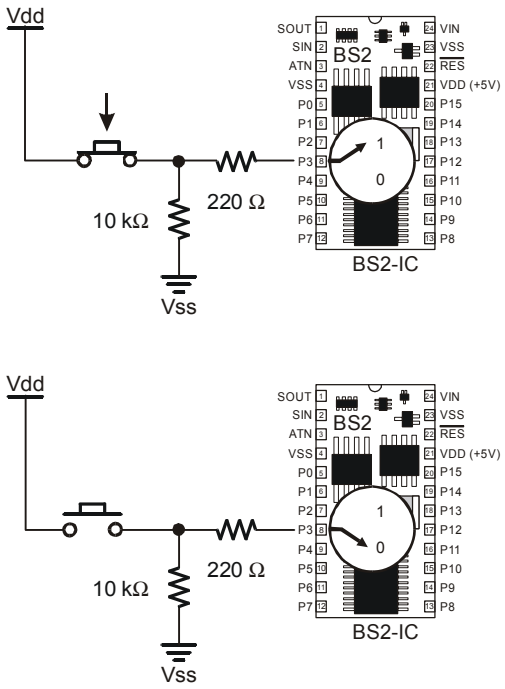


Abbildung 3-7
BASIC Stamp fragt Tasterzustand ab

Wird der Taster gedrückt, erkennt die BASIC Stamp eine 1 (oben). Wird der Taster nicht gedrückt, erkennt die BASIC Stamp eine 0 (unten).

Binäre Zustände in Schaltungen: Das Binärsystem kennt nur die Zustände 0 und 1. Diese Zustände lassen sich von einem Gerät auf ein zweites übertragen. Die BASIC Stamp interpretiert Vdd (5 V) als binäre 1 und Vss (0 V) als binäre 0. Wenn die BASIC Stamp einen I/O Pin mit **`HIGH`** auf Vdd setzt, sendet sie eine binäre 1. Wird ein I/O Pin mit **`LOW`** auf Vss gesetzt, wird eine binäre 0 übertragen. Dies ist ein üblicher Weg, mit dem binäre Zahlen zwischen Computern und anderen Geräten ausgetauscht werden.

Taster mit BASIC Stamp überwachen

Die BASIC Stamp speichert den am I/O Pin P3 festgestellten Zustand in einem Speicher mit der Bezeichnung **`IN3`**. In einem Beispielprogramm zeigen wir, was genau passiert:

Programmbeispiel: ReadPushbuttonState.bs2

In diesem Programm fragt die BASIC Stamp alle 250 ms den Taster ab und sendet den festgestellten Wert von **`IN3`** an das *Debug Terminal*. Abbildung 3-8 zeigt das *Debug*

Terminal während eines Programmlaufs. Wird der Taster gedrückt, erscheint eine 1, bleibt der Taster unberührt, erscheint eine 0.

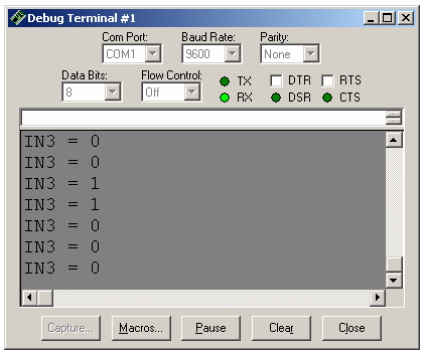


Abbildung 3-8
Das Debug Terminal zeigt die Zustände des Tasters an.

Das Debug Terminal zeigt eine 1, wenn der Taster gedrückt wird und eine 0, wenn er unberührt bleibt.

- √ Gib das Programm ReadPushbuttonState.bs2 in den BASIC Stamp Editor ein.
- √ Führe das Programm aus.
- √ Überprüfe, ob das Debug Terminal 0 anzeigt, wenn der Taster unberührt bleibt.
- √ Überzeuge dich, das das Debug Terminal eine 1 anzeigt, sobald der Taster gedrückt wird.

```
' What's a Microcontroller - ReadPushbuttonState.bs2
' Check and send pushbutton state to Debug Terminal every 1/4 second.

' {$STAMP BS2}
' {$PBASIC 2.5}

DO

  DEBUG ? IN3
  PAUSE 250

LOOP
```

Arbeitsweise von ReadPushbuttonState.bs2

Die Schleife **DO…LOOP** wird alle 250 ms wiederholt (der Befehl **PAUSE 250** erzwingt eine Pause von 250 ms). Mit **DEBUG ? IN3** wird bei jedem Schleifendurchlauf der Wert von **IN3** an das Debug Terminal gesendet. Der Wert von **IN3** gibt den Zustand des I/O Pins P3 zum Zeitpunkt der Befehlsausführung von **DEBUG** wieder.

Du bist dran – Taster mit Pull-up Widerstand

Im Schaltkreis der letzten Übung wurde ein Widerstand eingesetzt, der mit Vss verbunden war. Dieser Widerstand wird auch als Pull-down Widerstand bezeichnet, da er die Spannung an P3 auf 0 V zieht, solange der Taster unberührt bleibt. Abbildung 3-9 zeigt eine Tasterschaltung, die einen Pull-up Widerstand benutzt. Dieser Widerstand zieht die Spannung auf Vdd (5 V), solange der Taster unberührt bleibt und speichert in `IN3` eine 1 ab. Wird der Taster gedrückt liegt in `IN3` eine 0.

Der 220 Ω Widerstand wird in dieser Beispielschaltung als Schutzwiderstand für die I/O Pins der BASIC Stamp benötigt. In kommerziellen Produkten wird ein solcher Widerstand aus Kostengründen häufig durch eine Drahtbrücke ersetzt.

- √ Änder die Schaltung, wie in Abbildung 3-9 gezeigt, ab.
- √ Starte erneut ReadPushbuttonState.bs2.
- √ Überprüfe mit Hilfe des Debug Terminals ob `IN3` die 1 anzeigt, wenn der Taster nicht gedrückt ist und eine 0, sobald der Taster gedrückt wird.

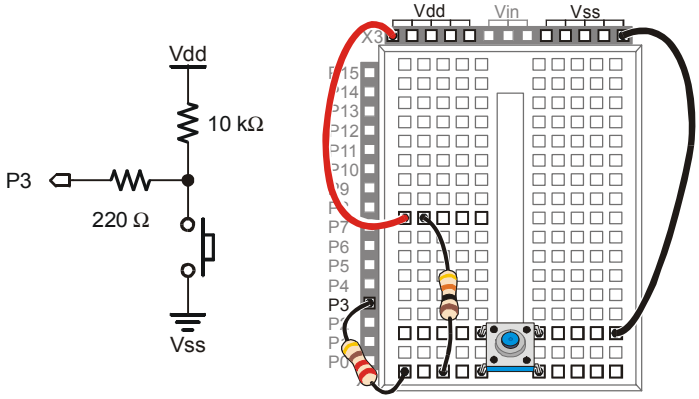


Abbildung 3-9
Abgeänderte Schaltung mit Taster

Aktiv-LOW vs. Aktiv-HIGH: Die Tasterschaltung in Abbildung 3-9 wird als aktiv-LOW bezeichnet, da sie an die BASIC Stamp ein LOW-Signal (Vss) überträgt, wenn der Taster aktiviert (gedrückt) wird. Die Schaltung mit Taster aus Abbildung 3-6 ist aktiv-HIGH, da sie ein HIGH-Signal (Vdd) sendet, wenn der Schalter aktiviert (gedrückt) wird.

ÜBUNG 3: TASTERSTEUERUNG EINER LED SCHALTUNG

Abbildung 3-10 zeigt die vergrößerte Abbildung eines Schalters und einer LED, wie man sie zum Beispiel an einem Computermonitor finden kann. Es gibt viele Beispiele von Geräten, die mit einem Taster ausgestattet sind und deren LED etwas über den Zustand des Gerätes aussagt.

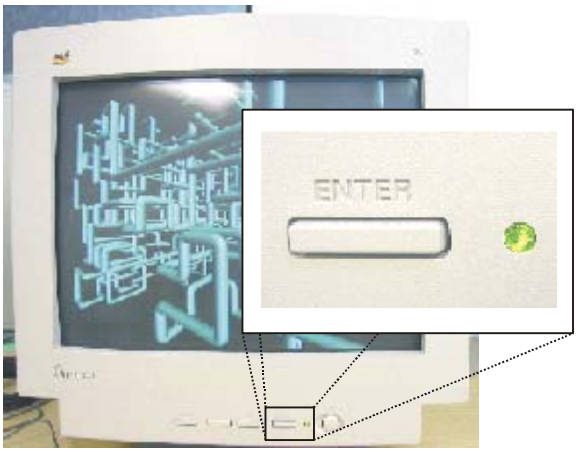


Abbildung 3-10
Schalter mit LED
an einem
Computermonitor

Die BASIC Stamp lässt sich so programmieren, dass sie Entscheidungen in Abhängigkeit von einem registrierten Zustand trifft. Sie lässt sich beispielsweise so einstellen, dass bei einem Tastendruck die angeschlossene LED zehn mal an und aus geht.

Bauteile: Taster und LED Schaltung

1 Taster - Schließer
1 Widerstand – 10 kΩ (braun-schwarz-orange)
1 LED – beliebige Farbe
1 Widerstand – 220 Ω (rot-rot-braun)
1 Widerstand – 470 Ω (gelb-violett-braun)
2 Verbindungsdrähte

Schaltungsaufbau mit Taster und LED

Abbildung 3-11 zeigt die Schaltung mit Taster aus Kapitel2, Übung 2.

√ Baue die Schaltung nach Abbildung 3-11 auf.

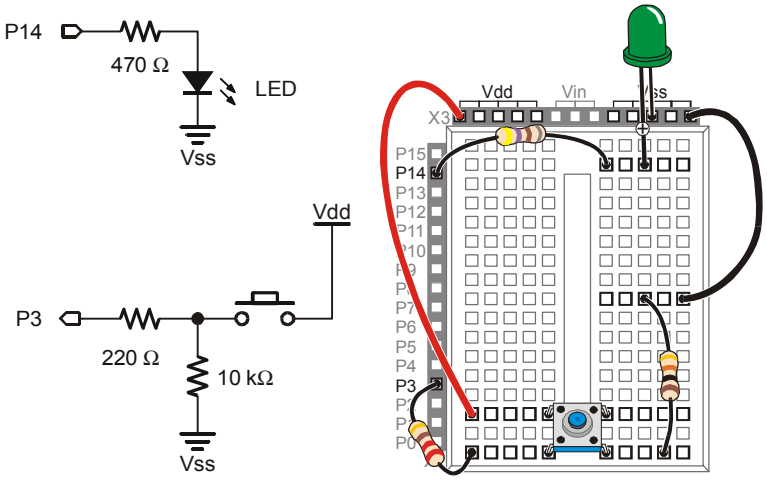


Abbildung 3-11
Schaltung mit Taster und LED

Programmierung eines Tasters

Die BASIC Stamp kann über eine **IF…THEN…ELSE** Anweisung so programmiert werden, dass sie Entscheidungen trifft. Das folgende BASIC Programm wird zum Beispiel eine LED ein- und ausschalten, sobald der Taster gedrückt wird. Jedesmal, wenn die **DO…LOOP** Schleife durchlaufen wird, überprüft die **IF…THEN…ELSE** Anweisung, ob der Taster gedrückt ist und entscheidet dann, ob die LED blinkt oder ausgeschaltet bleibt.

Programmbeispiel: PushbuttonControlledLed.bs2

√ Gib das Programm PushbuttonControlledLed.bs2 in den BASIC Stamp Editor ein und führe es anschließend aus.
√ Überprüfe, ob die LED blinkt, solange der Taster gedrückt wird.
√ Überprüfe, ob die LED aus bleibt, solange der Taster nicht gedrückt ist.

```
' What's a Microcontroller - PushbuttonControlledLed.bs2
' Check pushbutton state 10 times per second and blink LED when pressed.

' {$STAMP BS2}
' {$PBASIC 2.5}

DO
```

```
  DEBUG ? IN3

  IF (IN3 = 1) THEN
    HIGH 14
    PAUSE 50
    LOW 14
    PAUSE 50

  ELSE
    PAUSE 100

  ENDIF

LOOP
```

Arbeitsweise von PushbuttonControlledLed.bs2

Dieses Programm ist eine veränderte Programmversion von ReadPushbuttonState.bs2 aus der letzten Übung. Die Befehle **`DO…LOOP`** und **`DEBUG ? IN3`** wurden nicht verändert; **`PAUSE 250`** wurde durch die Anweisung **`IF…THEN…ELSE`** ausgetauscht. Wenn die Bedingung der **`IF`** Anweisung wahr ist **`(IN3 = 1)`**, werden alle Befehle nach der **`THEN`** Anweisung bis zum folgenden **`ELSE`** Befehl ausgeführt; anschließend springt das Programm in die Zeile mit **`ENDIF`** und fährt im Programm weiter fort. Ist die Aussage der Bedingung nach der **`IF`** Anweisung nicht wahr **`(IN3 = 0)`**, werden alle Befehle nach der **`ELSE`** Anweisung bis zum Befehlsaufruf **`ENDIF`** ausgeführt.

Eine ausführliche Liste, die beschreibt, was ein Programm können sollte, kann für die eigene Programmplanung und/oder eine Programmbeschreibung sehr hilfreich sein. Eine solche Liste bezeichnen wir als Pseudocode. In dem unten dargestellten Beispiel wird mit Hilfe eines Pseudocodes die Arbeitsweise des Programms PushbuttonControlledLed.bs2 beschrieben.

- *Führe alle Befehle zwischen hier und dem Schleifenbefehl immer wieder aus*
  - *Gib den Wert von IN3 auf dem Debug Terminal aus*
  - *Ist IN3 identisch 1, dann*
    - *Schalte die LED an*
    - *Warte 1/20 Sekunde*
    - *Schalte die LED aus*
    - *Warte 1/20 Sekunde*
  - *Sonst, (wenn IN3 identisch 0)*
    - *Unternimm nichts, warte 1/10 Sekunde.*
- *Schleifenende*

Du bist dran – Schneller/Langsamer

- √ Speicher das Beispielprogramm unter einem anderen Namen.
- √ Verändere das Programm so, dass die LED doppelt so schnell blinkt, wenn der Taster gedrückt wird.
- √ Verändere das Programm so, dass die LED nur noch halb so schnell wie zu Beginn dieser Übung blinkt.

ÜBUNG 4: ZWEI TASTER STEUERN ZWEI LED SCHALTUNGEN

Im folgenden Projekt wird ein zweiter Taster in die Schaltung eingebaut und die Wirkungsweise untersucht. Wir machen es noch etwas komplizierter und fügen auch eine zweite LED Schaltung hinzu, die von dem zweiten Taster gesteuert wird.

Schaltungsbauteile der LED Schaltungen mit Taster

2 Taster als Schließer
2 Widerstände - 10 kΩ (braun-schwarz-orange)
2 Widerstände – 470 Ω (gelb-violett-braun)
2 Widerstände – 220 Ω (rot-rot-braun)
2 LEDs – beliebige Farbe

Schaltungserweiterung mit Taster und LED

Abbildung 3-12 zeigt eine zweite Schaltung mit LED und Taster, die zur Schaltung aus der vorherigen Übung hinzugefügt wurde.

- √ Baue die Schaltung nach Abbildung 3-12 auf. Ist Hilfe erforderlich, schaue auf den Verdrahtungsplan in Abbildung 3-13 .
- √ Änder das Programm ReadPushbuttonState.bs2 so ab, dass es `IN4` statt `IN3` ausliest und teste die zweite Schaltung.

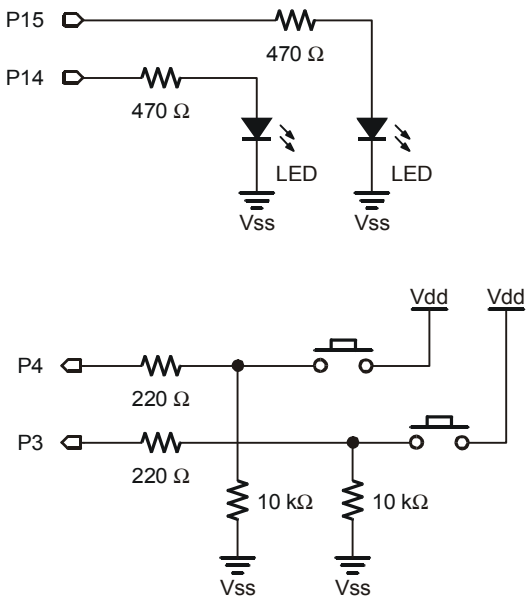


Abbildung 3-12
Schaltungsaufbau: Zwei Schalter und LEDs

Schaltungspunkte: Es gibt in Abbildung 3-12 drei Stellen, an denen sich mehrere Anschlussdrähte kreuzen aber nur zwei Schaltungspunkte. Schnittpunkte, die mit einem Punkt markiert sind, zeigen an, dass die Drähte dort zusammen laufen und Kontakt miteinander haben. Der Draht, der den P4 Schalter mit dem 10 kΩ Widerstand verbindet, hat keine Verbindung mit dem P3 Taster des zweiten Schaltkreises, da der Schnittpunkt der Drähte dort keinen Schaltungspunkt ausweist.

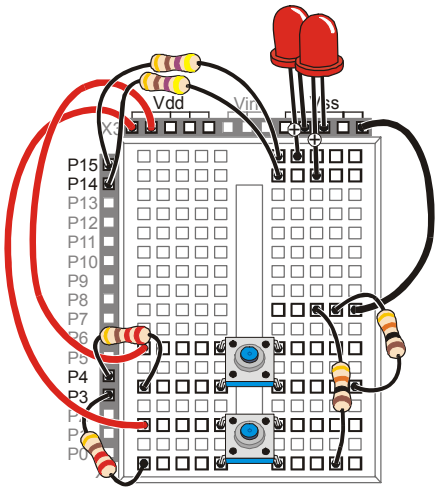


Abbildung 3-13
Verdrahtungsaufbau: Zwei Taster und LEDs

Programmierung der Tastersteuerung

In der vorherigen Übung haben wir mit einem Programm experimentiert, das mit der **IF…THEN…ELSE** Anweisung selbst Entscheidungen treffen konnte. Im folgenden Beispiel schauen wir uns einen neuen Entscheidungsaufruf an, der mit der **IF…ELSEIF…ELSE** Anweisung arbeitet. Mit ihm entscheiden wir, welche LED in welcher Schaltung ein- bzw. ausgeschaltet wird.

Programmbeispiel: PushbuttonControlOfTwoLeds.bs2

- √ Gib das Programm PushbuttonControlOfTwoLeds.bs2 in den BASIC Stamp Editor ein und führe es aus.
- √ Überprüfe, ob die mit P14 verbundene LED blinkt, solange der Taster an P3 gedrückt bleibt.
- √ Überprüfe ebenso, ob die mit P15 verbundene LED blinkt, solange der Taster an P4 gedrückt bleibt.

```
' What's a Microcontroller - PushbuttonControlOfTwoLeds.bs2
' Blink P14 LED if P3 pushbutton is pressed, and blink P15 LED if
' P4 pushbutton is pressed.

' {$STAMP BS2}
' {$PBASIC 2.5}

DO
```

```
  DEBUG HOME
  DEBUG ? IN4
  DEBUG ? IN3

  IF (IN3 = 1) THEN
    HIGH 14
    PAUSE 50

  ELSEIF (IN4 = 1) THEN
    HIGH 15
    PAUSE 50

  ELSE
    PAUSE 50

  ENDIF

  LOW 14
  LOW 15

  PAUSE 50

LOOP
```

Arbeitsweise von PushbuttonControlOfTwoLeds.bs2

Wenn die Werte von **IN3** und **IN4** auf dem Debug Terminal, wie im vorherigen Beispiel geschehen, ausgegeben werden und gleich darauf - weil dort weitere Ausgaben erfolgen – wieder verschwinden, wird das Ablesen schwierig. Um dies zu verhindern, sollten Ausgaben mit dem Formatierungsbefehl **HOME** immer oben links im Debug Terminal beginnen:

```
DEBUG HOME
```

Der Cursor gibt jetzt bei jedem Schleifendurchlauf von **DO…LOOP** mit den Befehlen:

```
DEBUG ? IN4
DEBUG ? IN3
```

die Werte von **IN4** und **IN3** an derselben Stelle des Debug Terminals aus.

Mit dem Schlüsselwort **DO** beginnt die Schleife des Programms:

```
DO
```

Die Befehle nach der **IF** Anweisung sind die gleichen wie im Beispiel der vorherigen Übung:

```
IF (IN3 = 1) THEN
  HIGH 14
  PAUSE 50
```

Hier hilft das Schlüsselwort **ELSEIF**. Wenn **IN3** ungleich 1, aber **IN4** gleich 1 ist, soll die LED an P15 statt an P14 eingeschaltet werden.


```
ELSEIF (IN4 = 1) THEN
  HIGH 15
  PAUSE 50
```

Wenn keine der Anweisungen erfüllt ist, soll das Programm für 50 ms eine Pause einlegen, ohne dass dabei der Zustand der LEDs verändert wird.

```
ELSE
  PAUSE 50
```

Sind alle Entscheidungen abgearbeitet, vergiss bitte nicht den Befehl **ENDIF** zu setzen.

```
ENDIF
```

Jetzt sollten die LEDs ausgeschaltet und eine Pause eingelegt werden. Egal in welchem Zustand sie sich gerade befinden, wir fragen ihn nicht ab, sondern schalten beide LEDs einfach ab.

```
LOW 14
LOW 15

PAUSE 50
```

Die **LOOP** Anweisung schickt das Programm zurück zur **DO** Anweisung und der oben beschriebene Prozess wiederholt sich beliebig oft.

```
LOOP
```

Du bist dran – Beide Taster gleichzeitig drücken

Das Beispielprogramm hat eine Schwachstelle: versuche einmal beide Taster möglichst gleichzeitig zu drücken. Wenn du angenommen hast, dass jetzt beide LEDs anfangen zu blinken, wirst du enttäuscht. Es passiert nicht, weil nur ein Entscheidungsblock innerhalb einer **IF...ELSEIF…ELSE** Anweisung ausgeführt wird und das Programm anschließend auf **ENDIF** springt.
Es folgt ein Lösungsvorschlag:

√ Speicher das Programm PushbuttonControlOfTwoLeds.bs2 unter einem neuen Namen ab.

√ Ersetze die **IF** Anweisung und die Programmzeilen:

```
IF (IN3 = 1) THEN
  HIGH 14
  PAUSE 50
```

durch die **IF...ELSEIF** Anweisung:

```
IF (IN3 = 1) AND (IN4 = 1) THEN
  HIGH 14
  HIGH 15
  PAUSE 50

ELSEIF (IN3 = 1) THEN
  HIGH 14
  PAUSE 50
```

Ein Programmblock fasst eine Gruppe von Befehlen zusammen. Die oben dargestellte **IF** Anweisung besteht aus einem Programmblock mit drei Befehlen (**HIGH**, **HIGH** und **PAUSE**). Die **ELSEIF** Anweisung enthält einen Programmblock aus zwei Befehlen (**HIGH**, **PAUSE**).

√ Führe das veränderte Programm aus und beobachte, ob es wie erwartet reagiert.

Das Schlüsselwort AND wird in einer **IF…THEN** Anweisung eingesetzt, um überprüfen zu können, ob mehr als eine Bedingung wahr ist. Eine **IF** Anweisung ist nur dann wahr, wenn alle mit **AND** verknüpften Bedingungen wahr sind.

Das Schlüsselwort OR wird dann benutzt, wenn überprüft werden soll, ob mindestens eine Bedingung wahr ist.

Durch Änderungen am Programm kann auch die Blinkzeit der LED verändert werden. Reduziere dazu einmal das Argument ***Duration*** im Befehl **PAUSE** bei beiden Tastern auf 10, erhöhe **PAUSE** für die LED an P14 auf 100 und erhöhe für die LED an P15 **PAUSE** auf 200.

√ Änder den Befehl **PAUSE** in der **IF** und den beiden **ELSEIF** Anweisungen wie oben besprochen.
√ Führe das abgeänderte Programm aus.
√ Beobachte den Unterschied im Verhalten der beiden Lichtquellen.

ÜBUNG 5: BAU EINES REAKTIONSZEITMESSERS

Als Entwicklungsingenieur einer Firma für Videospiele möchte die Marketingabteilung für ihre neue Spielekonsole eine Schaltung entwickeln lassen, mit der ein Spieler auch seine Reaktionszeit testen kann. Deine Aufgabe ist es nun, eine Schaltung zu entwerfen, mit der sich die Reaktionszeit bestimmen lässt.

Die Lösung für einen möglichen Schaltungsentwurf wird in dieser Übung aufgebaut und getestet. Es ist eine von vielen Möglichkeiten und soll hier beispielhaft vorgestellt werden. Bevor du weiter liest, halte einen Moment inne und überlege, wie dein Reaktionstester aussehen würde und wie du das Problem lösen würdest.

Bauteile: Reaktionszeitmesser

1 DUO LED
1 Widerstand – 470 Ω (gelbt-violett-braun)
1 Taster als Schließer
1 Widerstand – 10 kΩ (braun-schwarz-orange)
1 Widerstand – 220 Ω (rot-rot-braun)
2 Verbindungsdrähte

Schaltungsaufbau Reaktionszeitmesser

Abbildung 3-14 zeigt den Schaltungsaufbau und den Verdrahtungsplan einer Schaltung mit der BASIC Stamp zur Feststellung der Reaktionszeit.

- √ Baue die Schaltung nach Abbildung 3-14 auf.
- √ Führe TestBiColorLED.bs2 aus Kapitel2, Übung 5 aus und teste die DUO LED Schaltung.
- √ Baue die Schaltung mit Taster noch einmal auf, starte das Programm ReadPushbuttonState.bs2 von Übung 2 aus diesem Kapitel und überprüfe, ob der Taster fehlerfrei arbeitet.

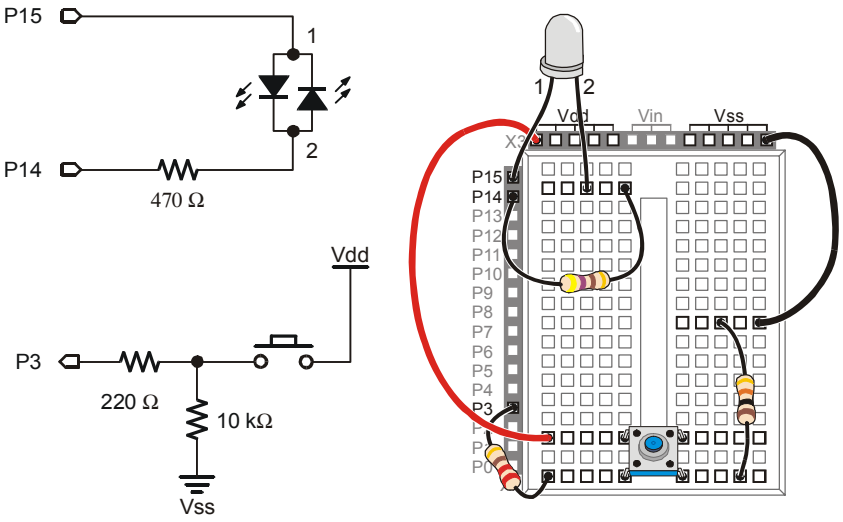


Abbildung 3-14
Schaltung für Reaktionszeit-messung

Programmierung des Reaktionszeitmessers

In diesem Beispielprogramm bleibt die DUO LED so lange ausgeschaltet, bis der Spieler den Taster drückt. Sobald er gedrückt wird, leuchtet die LED für eine kurze Zeit rot auf und springt schließlich auf grün. Die ist das Startsignal für den Spieler, den Taster möglichst schnell loszulassen. Die Zeit, die zwischen dem Umschalten der LED auf grün und dem Loslassen des Tasters vergeht (d.h. die Zeit, bis das Programm dies festgestellt hat), wird als Maß für die Reaktionszeit genommen.

Das Beispielprogramm zeigt wie Abfrage und Zählvorgang funktionieren. Die Abfrage beschreibt einen Vorgang, bei dem ein Gerät, einen Eingangskanal permanent auf Änderung überprüft. Bei einem Zählvorgang wird eine vorgegebene Zahl jedes Mal, wenn sich etwas verändert oder nicht verändert hat (die Bedingung wird vorher festgelegt), um einen bestimmten Betrag erhöht oder erniedrigt. In diesem Programm wird die BASIC Stamp über einen Zählvorgang die Zeit erfassen, die vom Umschalten der DUO LED auf grün bis zum Loslassen des Tasters vergangen ist. Mit dem Befehl **`PAUSE 1`** wird eine definierte Pause von 1/1000 Sekunde eingelegt. Jedesmal, wenn bei der Abfrage festgestellt wird, dass der Taster noch nicht losgelassen wurde, erhöht sich der Inhalt der Zählvariable **`timeCounter`** um 1. Sobald der Taster losgelassen wird, stoppt die Abfrage und auf dem Debug Terminal wird der Inhalt des Zählspeichers **`timeCounter`** ausgegeben.

Programmbeispiel: ReactionTimer.bs2

√ Gib das Programm ReactionTimer.bs2 ein und führe es anschließend aus.
√ Folge den Anweisungen des Debug Terminals (Abbildung 3-15).

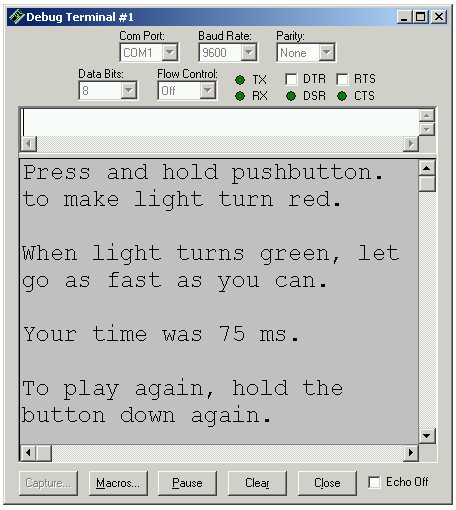


Abbildung 3-15
Debug Terminal
Anweisungen zum Spiel
Reaktionszeitmesser

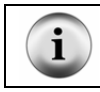
```
' What's a Microcontroller - ReactionTimer.bs2
' Test reaction time with a pushbutton and a bi-color LED.

' {$STAMP BS2}
' {$PBASIC 2.5}

timeCounter    VAR     Word                  ' Declare variable to store time.

DEBUG "Press and hold pushbutton.", CR,      ' Display reaction instructions.
      "to make light turn red.", CR, CR,
      "When light turns green, let", CR,
      "go as fast as you can.", CR, CR

DO                                           ' Begin main loop.

  DO                                         ' Nested loop repeats...
  LOOP UNTIL IN3 = 1                         ' until pushbutton press.

  LOW 14                                     ' Bi-color LED red.
  HIGH 15

  PAUSE 1000                                 ' Delay 1 second.
```

```
  HIGH 14                                     ' Bi-color LED green.
  LOW 15

  timeCounter = 0                             ' Set timeCounter to zero.

  DO                                          ' Nested loop, count time...

    PAUSE 1
    timeCounter = timeCounter + 1

  LOOP UNTIL IN3 = 0                          ' until pushbutton is released.

  LOW 14                                      ' Bi-color LED off.

  DEBUG "Your time was ", DEC timeCounter,    ' Display time measurement.
        " ms.", CR, CR,
        "To play again, hold the ", CR,       ' Play again instructions.
        "button down again.", CR, CR

LOOP                                          ' Back to "Begin main loop".
```

Arbeitsweise von ReactionTimer.bs2

Das Programm behält die Übersicht über alle Tastendrucke, deren Anzahl in der Variablen **`timeCounter`** abgelegt wurden.

```
timeCounter VAR Word          ' Declare variable to store time.
```



Variablen auf Null initialisieren: Wird in PBASIC eine Variable deklariert, wird ihr Wert automatisch auf Null gesetzt, bis mit einem weiteren Befehl der Wert verändert wird.

Der **`DEBUG`** Befehl enthält Anweisungen für den Spieler.

```
DEBUG "Press and hold pushbutton.", CR,
      "to make light turn red.", CR, CR,
      "When light turns green, let", CR,
      "go as fast as you can.", CR, CR
```

`DO…LOOP` Anweisungen können geschachtelt sein, d.h. eine **`DO…LOOP`** Schleife kann innerhalb einer anderen **`DO…LOOP`** Schleife aufgerufen werden.

```
DO                            ' Begin main loop.

  DO                          ' Nested loop repeats...
  LOOP UNTIL IN3 = 1          ' until pushbutton press.
```

```
  ' Rest of program was here.

LOOP                            ' Zurück zu "Begin main loop".
```


Schauen wir uns einmal die innere **DO…LOOP** Schleife genauer an. Bei einer **DO…LOOP** Anweisung kann mit Hilfe einer Bedingung entschieden werden, ob man die Schleife verlassen möchte und mit den Befehlen nach der Programmzeile **LOOP** weitermachen möchte oder nicht. Die innere Schleife wird, so lange der Taster nicht gedrückt wird (**IN3 = 0**), ständig durchlaufen. In dem Moment, wo der Taster gedrückt wird (**IN3 = 1),** fährt das Programm mit der Zeile, die nach der **LOOP UNTIL** Anweisung kommt, fort. Dies ist ein Beispiel für eine Datenabfrage. Mit Hilfe der Schleife **DO…LOOP UNTIL** wird solange der Zustand des Tasters überprüft, bis sich sein Zustand verändert (Taster wird gedrückt).

```
DO                              ' innere Schleife durchlaufen...
LOOP UNTIL IN3 = 1              ' bis Taster gedrückt wird.
```

Der Befehl unmittelbar im Anschluss an die **LOOP UNTIL** Anweisung schaltet die rot leuchtende DUO LED mit einer Verzögerungszeit von einer Sekunde auf grün.

```
LOW 14                           ' DUO LED rot.
HIGH 15

PAUSE 1000                       ' Verzögerung 1 Sekunde.

HIGH 14                          ' DUO LED grün.
LOW 15
```

Im Moment des Umschaltens von rot auf grün startet die Zeitmessung und läuft so lange, bis der Spieler den Taster losgelassen hat. Die Variable **timeCounter** ist auf Null initialisiert und die folgende **DO…LOOP** Schleife mit der Abbruchbedingung **UNTIL** wird so lange durchlaufen, bis der Spieler den Taster loslässt (**IN3 = 0**). Bei jedem Schleifendurchlauf hält die BASIC Stamp mit **PAUSE 1** für 1 ms an und erhöht den Zählwert der Variablen **timeCounter** um eins.

```
timeCounter = 0                  ' Set timeCounter to zero.

DO                               ' Innere Schleife misst Zeit...

  PAUSE 1
  timeCounter = timeCounter + 1  ' Zähler um eins erhöhen

LOOP UNTIL IN3 = 0               ' bis Taster losgelassen wird.
```

Die DUO LED wird abgeschaltet.

```
LOW 14
```

Die Ergebnisse erscheinen auf dem Debug Terminal.

```
DEBUG "Your time was ", DEC timeCounter,
      " ms.", CR, CR,
      "To play again, hold the ", CR,
      "button down again.", CR, CR
```

Die letzte Anweisung des Programmes ist **`LOOP;`** das Programm kehrt zur ersten **`DO`** Anweisung zurückkehrt.

Du bist dran – Das Programm überarbeiten

Der soeben entwickelte Prototyp eines Spiels wurde von der Marketingabteilung an Testspieler übergeben. Als Testergebnis wurde festgestellt, dass drei Punkte abgeändert werden müssen, bevor man das Spiel in eine Spielekonsole einbauen kann.

√ Speicher das Programm ReactionTimer.bs2 unter einem anderen Namen (z.B. ReactionTimerYourTurn.bs2) ab.

Folgende Problempunkte und mögliche Lösungen haben die Testspieler angegeben.

Punkt 1

Hält ein Spieler den Taster 30 Sekunden, werden tatsächlich 14000 ms oder 14 Sekunden angezeigt. Das muss korrigiert werden!

Es stellt sich heraus, dass für die Ausführung eines Schleifendurchlaufs und das Addieren von 1 zur Variablen **`timeCounter`** ungefähr 1 ms benötigt werden, ohne dass dabei der Befehl **`PAUSE 1`** berücksichtigt wurde. Das liegt daran, dass der Controller selbst zur Ausführung jedes Programmbefehls Zeit benötigt. Wir können das leicht überprüfen, indem wir den Befehl **`PAUSE 1`** auskommentieren (setze ein Apostroph an den Anfang der Zeile).

```
' PAUSE 1
```

√ Kommentiere PAUSE 1 aus und teste das Programm erneut. Wie zeitgenau arbeitet es jetzt?

Statt die Verzögerung auszukommentieren, ließe sich das Programm auch dadurch korrigieren, dass das Ergebnis mit zwei multipliziert wird, indem du vor die **DEBUG** Befehlszeile, die für die Zeitangabe (in ms) verantwortlich ist, einen Befehl einfügst, der das Ergebnis mit zwei multipliziert:

```
timeCounter = timeCounter * 2              ' <- Add this
DEBUG "Your time was ", DEC timeCounter, " ms.", CR, CR
```

√ Aktiviere den Befehl **PAUSE** durch Löschen des Apostrophs und überprüfe die Genauigkeit des neuen Programms (Multiplikation mit zwei).

Höhere Genauigkeit: mit Hilfe des Operators */ lassen sich auch Multiplikationen mit Dezimalzahlen ausführen. Wie der */ Operator aufgerufen wird, erfährst du hier:

1) Setze den Wert oder die Variable, die mit einer rationalen Zahl multiplizert werden soll, links vor den */ Operator.
2) Multipliziere als nächstes die rationale Zahl mit 256.
3) Runde die Zahl auf die Einerstelle.
4) Platziere den Wert rechts vor den */ Operator.

Beispiel: Der Zahlenwert in der Variablen timeCounter soll mit 3.69 multipliziert werden.

1) Setze timeCounter links vor den */ Operator:

   timeCounter = timeCounter */

2) Multipliziere die rationale Zahl mit 256: 3.69 X 256 = 944.64.
3) Runde auf: 944.64 ≈ 945.
4) Platziere den Wert rechts vor den */ Operator:

   timeCounter = timeCounter */ 945

Punkt 2

Spieler finden sehr schnell heraus, dass die Verzögerungszeit von rot auf grün eine Sekunde beträgt. Nach einigen Spielrunden werden sie die feste Verzögerungszeit abschätzen und darauf besser reagieren können; damit gibt die gemessene Verzögerungszeit nicht mehr die wahre Reaktionszeit eines Spielers an.

Die BASIC Stamp verfügt über einen **RANDOM** Befehl. Die folgenden Zeilen geben an, wie das Programm abgeändert werden muss, um Zufallszahlen aufzurufen:

√ Deklariere am Anfang des Programm eine neue Variable **`value`** und weise ihr den Wert 23 zu. 23 wird auch als Startwert der Zufallszahlensequenz bezeichnet.

```
timeCounter VAR Word
value VAR Byte                          ' <- Add this
value = 23                              ' <- Add this
```

√ Füge direkt vor dem Pausenbefehl den Befehl **`RANDOM`** ein und weise **`value`** einen neuen Zufallswert zu, der aus einer Pseudozufallssequenz, die mit 23 startet, erzeugt wird.

```
RANDOM value                            ' <- Add this.
DEBUG "Delay time ", ? 1000 + value, CR   ' <- Add this.
```

√ Veränder den Befehl **`PAUSE`** so, dass der Zufallszahlenwert im Argument ***`Duration`*** des Befehls **`PAUSE`** zu 1000 (entspricht einer Sekunde) addiert wird.

```
PAUSE 1000 + value                      ' <- Modify this.
```

Was ist ein Algorithmus ? Unter einem Algorithmus versteht man eine Folge mathematischer Operationen.

Was heißt Pseudozufall? Pseudozufall bedeutet, dass in Wirklichkeit kein Zufall vorliegt. Jedes mal, wenn das Programm neu startet, erzeugt es die gleiche Zahlensequenz.

Startwert? Mit dem Startwert wird der Beginn der Pseudo-Zufallszahlensequenz festgelegt. Wird ein anderer Startwert gewählt, ändert sich auch die Pseudo Zufallszahlensequenz.

Punkt 3

Ein Spieler, der den Taster loslässt, bevor die LED auf grün springt, kann damit seine Reaktionszeit unerlaubterweise deutlich verbessern. Der Mikrocontroller muss so programmiert sein, dass er betrügerisches Vorgehen von Spielern erkennt.

Am Ende von Übung 3 in diesem Kapitel haben wir uns mit Pseudocode beschäftigt. Der folgende Pseudocode gibt einen Hinweis, wie man mit Hilfe der Entscheidungsanweisung **`IF…THEN…ELSE`** das Problem lösen könnte.

- *Falls der Wert von timeCounter identisch 1 ist*

    - *Zeige dem Spieler an, dass er/sie so lange den Taster gedrückt halten und warten muss, bis die LED auf grün springt.*
- *Sonst, (falls der Wert von timeCounter größer 1 ist)*
    - *Zeige den Wert von timeCounter (so, wie bei ReactionTimer.bs2) an, er entspricht der Zeit in ms.*

√ Änder das Programm nach Vorlage des Pseudo Codes in PBASIC so ab, dass das oben dargestellte Spielerproblem behoben wird.

ZUSAMMENFASSUNG

In diesem Kapitel hast du Taster und Schaltkreise mit Tastern kennengelernt. Wir haben erfahren, wie man Schaltungen mit Tastern aufbaut und prüft und die BASIC Stamp einsetzt, um den Zustand eines oder mehrerer Taster abzufragen. Die BASIC Stamp steuerte LEDs über Entscheidungen, die sich aus den Schaltzuständen der angeschlossenen Taster ergaben. Aus den vorgestellten Programmkonzepten wurde ein Reaktionszeitspiel entwickelt. Eine LED wurde angesteuert und die BASIC Stamp leitete aus der Überprüfung des Tasterzustands eine Zeitmessung ab.

Ebenso wurde gezeigt, wie man beliebige Schaltungen mit Tastern über die speziellen in der BASIC Stamp verfügbaren I/O Variablen (**IN3**, **IN4**, etc.) anspricht. Dazu haben wir die eingelesenen Werte in Entscheidungsanweisungen **IF…THEN…ELSE** und **IF…ELSEIF…ELSE** eingebunden und erfahren, was unter dem Begriff Programmblock zu verstehen ist. Mit Hilfe der Operatoren **AND** und **OR** konnten zusammengesetzte Bedingungen ausgewertet werden. Wir haben ebenso mit der Anweisung **DO…LOOP** gearbeitet, Abbruchbedingungen über das Schlüsselwort **UNTIL** formuliert und verschachtelte Schleifen aufgebaut.

Verständnisfragen

1. Erkläre am Beispiel einer BASIC Stamp den Unterschied zwischen Senden und Empfangen von **HIGH** und **LOW** Signalen.
2. Was ist gemeint, wenn man einen Taster als Schließer bezeichnet?
3. Was passiert zwischen den Kontakten eines Tasters, wenn er gedrückt wird?
4. Welchen Wert nimmt **IN3** an, wenn ein Taster ihn mit Vdd verbindet? Welchen Wert nimmt **IN3** an, wenn ein Taster ihn mit Vss verbindet?
5. Was bewirkt der Befehl **DEBUG ? IN3** ?
6. Wie könnte eine Verzweigungsstruktur in einem Programm aussehen, die auf den Zustand eines oder mehrerer Taster reagiert?
7. Was bewirkt die Formatierungsanweisung **HOME** in der Anweisung **DEBUG HOME**?

Aufgaben

1. Erkläre, wie das Programm ReadPushbuttonState.bs2 auf Seite 81 abgeändert werden muss, damit der Taster jede Sekunde statt alle 250 ms abgefragt wird.

2. Erläuter, wie das Programm ReadPushbuttonState.bs2 abgeändert werden muss, damit es den Zustand einer Schaltung mit Schließer mit einem Pull-up Widerstand an I/O Pin P6 abfragen kann.

Projekt

1. Änder das Programm ReactionTimer.bs2 so ab, dass es ein Spiel für zwei Teilnehmer wird. Füge einen zweiten Taster an P4 hinzu.

Lösungen

Q1. Ausgehende Signale benutzen den BASIC Stamp I/O Pin als Ausgang, während eingehende Signale den I/O Pin als Eingang nutzen.
Q2. Im allgemeinen bezeichnet ein Schließer einen Taster, der im nicht gedrückten Zustand einen Schaltkreis unterbricht.
Q3. Wird der Taster gedrückt, werden die beiden Kontakte kurzgeschlossen und der Strom kann durch den Taster fließen.
Q4. **IN3** = 1 sobald der Taster ihn mit Vdd verbunden hat. **IN3** = 0 , sobald der Taster ihn mit Vss verbunden hat.
Q5. **DEBUG ? IN3** überträgt den Wert von **IN3** an das Debug Terminal.
Q6. **IF...THEN...ELSE** und **IF...ELSEIF...ELSE.**
Q7. Die Formatierungsanweisung **HOME** bewegt den Cursor in die obere linke Ecke des Debug Terminals.
E1. Die **DO...LOOP** Schleife des Programms wird alle 250 ms durchlaufen, da **PAUSE 250** gesetzt wurde. Um die Abtastzeit auf eine Sekunde zu setzen, setze **PAUSE 250** (250ms = 0.25 s = ¼ s) auf **PAUSE 1000** (1000ms = 1 s).

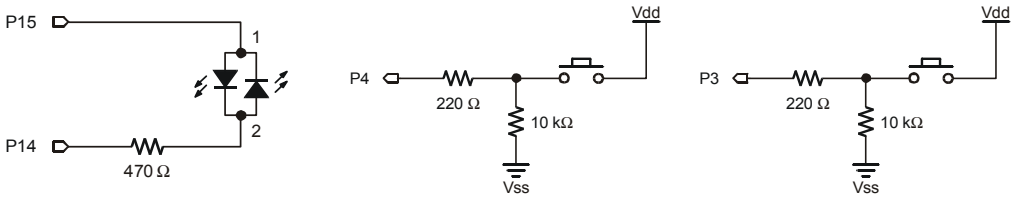
```
DO
   DEBUG ? IN3
   PAUSE 1000
LOOP
```

E2. Ersetze **IN3** mit **IN6** um I/O Pin P6 einzulesen. Das Programm gibt nur den Tasterzustand an, nutzt den Wert aber nicht, um daraus Entscheidungen abzuleiten; es unterscheidet nicht zwischen Pull-up und Pull-down Widerstand. Die **DEBUG** Anweisung zeigt den Tasterzustand in jedem Fall an.

```
DO
  DEBUG ? IN6
  PAUSE 250
```

```
LOOP
```

P1. Erst wird ein Taster für den zweiten Spieler an Stamp I/O Pin P4 hinzugefügt. Die Schaltskizze basiert auf Abbildung 3-14 von Seite 94.



Auszüge einer möglichen Programmlösung folgen. Die meisten Lösungen werden folgende Änderungszeilen enthalten:

Definiere zwei Variable für die Zeiterfassung.

```
timeCounterA VAR    Word      ' Time score of player A
timeCounterB VAR    Word      ' Time score of player B
```

Berücksichtige in den Anweisungen zwei Taster:

```
DEBUG "Press and hold pushbuttons", CR,
DEBUG "buttons down again.", CR, CR
```

Setze die LED erst dann auf rot, wenn beide Taster gedrückt sind; benutze dazu den **AND** Operator.

```
LOOP UNTIL (IN3 = 1) AND (IN4 = 1)
```

Beende die Zeitmessung erst dann, wenn beide Taster losgelassen sind. Benutze auch hier wieder den **AND** Operator.

```
LOOP UNTIL (IN3 = 0) AND (IN4 = 0)
```

Ergänze die Entscheidungslogik so, dass entschieden werden kann, bei welchem Spieler die Zeit erhöht werden muss.

```
IF (IN3 = 1) THEN
timeCounterA = timeCounterA + 1
ENDIF
IF (IN4 = 1) THEN
timeCounterB = timeCounterB + 1
ENDIF
```

Änder die Zeitdarstellung so, dass die Zeiten beider Spieler angegeben werden.

```
DEBUG "Player A Time:  ", DEC timeCounterA, " ms. ", CR
      DEBUG "Player B Time:  ", DEC timeCounterB, " ms. ", CR, CR
```

Füge einen Entscheidungsblock hinzu, der angibt, welcher Spieler die schnellste Reaktionszeit aufweist.

```
IF (timeCounterA < timeCounterB) THEN
DEBUG "Player A is the winner!", CR
ELSEIF (timeCounterB < timeCounterA) THEN
DEBUG "Player B is the winner!", CR
ELSE
DEBUG "It's a tie!", CR
ENDIF
```

Die vollständige Lösung ist unten dargestellt.

```
' What's a Microcontroller - Ch03Prj03_TwoPlayerReactionTimer.bs2
' Test reaction time with a pushbutton and a bi-color LED.
' Add a second player with a second pushbutton. Both players
' play at once using the same LED.  Quickest to release wins.
' Pin P3: Player A Pushbutton, Active High
' Pin P4: Player B Pushbutton, Active High

' {$STAMP BS2}
' {$PBASIC 2.5}

timeCounterA VAR        Word                ' Time score of player A
timeCounterB VAR        Word                ' Time score of player B

DEBUG "Press and hold pushbuttons", CR,     ' Display reaction
                                            ' instructions.
      "to make light turn red.", CR, CR,
      "When light turns green, let", CR,
      "go as fast as you can.", CR, CR

DO                                          ' Begin main loop.

  DO                                        ' Loop until both press
  ' Nothing
  LOOP UNTIL (IN3 = 1) AND (IN4 = 1)

  LOW 14                                    ' Bi-color LED red.
  HIGH 15

  PAUSE 1000                                ' Delay 1 second.

  HIGH 14                                   ' Bi-color LED green.
```

```
  LOW 15

  timeCounterA = 0                            ' Set timeCounters to zero
  timeCounterB = 0

  DO

    PAUSE 1
    IF (IN3 = 1) THEN                         ' If button is still down,
      timeCounterA = timeCounterA + 1         ' increment counter
    ENDIF
    IF (IN4 = 1) THEN
      timeCounterB = timeCounterB + 1
    ENDIF

  LOOP UNTIL (IN3 = 0) AND (IN4 = 0)          ' Loop until both buttons
                                              ' released.

  LOW 14                                      ' Bi-color LED off.

  DEBUG "Player A Time:  ", DEC timeCounterA, " ms. ", CR
  DEBUG "Player B Time:  ", DEC timeCounterB, " ms. ", CR, CR
  IF (timeCounterA < timeCounterB) THEN
    DEBUG "Player A is the winner!", CR
  ELSEIF (timeCounterB < timeCounterA) THEN
    DEBUG "Player B is the winner!", CR
  ELSE                                        ' A & B times are equal
    DEBUG "It's a tie!", CR
  ENDIF
  DEBUG CR

  DEBUG "To play again, hold the ", CR        ' Play again instructions.
  DEBUG "buttons down again.", CR, CR



LOOP                                          ' Back to "Begin main
```

Weiterführende Bücher

Die folgenden Bücher können kostenlos von der Parallax Website geladen werden und sind auch auf der Parallax CD zu finden.

***"BASIC Stamp Manual"*, User Manual, Version 2.0c, Parallax Inc., 2000**

In dem *BASIC Stamp Manual* befinden sich weitere Beispiele zum ausprobieren und zusätzliche Informationen, die die Befehle: `HIGH`, `LOW`, `PAUSE`, die Formatierungsanweisung `DEBUG ?` und `FOR…NEXT` näher erläutern.

"*Basic Analog and Digital*", Student Guide, Version 2.0, Parallax Inc., 2003

Basic Analog and Digital arbeitet mit LEDs und beschreibt binäre Zähler, analoge Abläufe und weist neue Wege auf, die Helligkeit von LEDs anzupassen.

"*BASIC Stamp Editor Help File*", PBASIC 2.5 Version 2.0 Parallax Inc., 2003

Die Hilfedatei von PBASIC 2.5 enthält Hinweise zur `DO…LOOP` Schleife, die neu in PBASIC 2.5 aufgenommen wurde und nicht im BASIC Stamp Manual enthalten ist. Klicke auf das Buchsymbol in der Task Bar deines BASIC Stamp Editor und wähle PBASIC Reference aus dem Menü im linken Fenster. Damit wird die alphabetisch sortierte PBASIC Befehlsreferenzliste geoffnet. Hier gibt es umfassende Informationen zu jedem Befehl.

Kapitel 4: Servosteuerung

BEWEGUNG MIT MIKROCONTROLLER STEUERN

Mikrocontroller sorgen Tag für Tag dafür, dass überall auf der Welt Dinge so bewegt werden, dass sie ihren Zielort sicher erreichen. Bei einem Tintenstrahldrucker bewegt sich der Druckkopf während eines Ausdrucks ständig von rechts nach links und zurück und wird über einen Schrittmotor, der von einem Controller angetrieben wird, gesteuert. Die Automatiktüren in Kaufhäusern werden ebenso wie die automatische Ausgabe einer Videocassette bei einem Videorecorder oder DVD-Spieler von einem Mikrocontroller gesteuert.

MOTORBEWEGUNG UND ON/OFF SIGNALE

Alle computergesteuerten Motoren werden über Serien kurzer HIGH/LOW Sequenzen, ähnlich wie wir es bei den LEDs kennengelernt haben, gesteuert. Die Zeitabfolge der Signale ist abhängig vom Motorenmodell und muss sehr schnell erfolgen, so dass sie mit bloßem Auge nicht mehr aufgelöst werden kann und sie erfordert einen schnellen Rechner, der in der Lage ist, diese Impulsserien zu liefern.

Einige dieser Motoren benötigen einen hohen Schaltungsaufwand um sie mit Hilfe eines Mikrocontrollers anzusteuern, während andere einen hohen mechanischen Aufwand erfordern, um sie in Geräten betreiben zu können. Der in diesem Kapitel vorgestellte Servomotor oder kurz Servo ist ein sehr einfaches Modell, das mit wenig Schaltungsaufwand mit der BASIC Stamp angesteuert und ohne große Mühe auch in Geräten eingebaut werden kann, um damit z.B. mechanische Teile zu bewegen.

ÜBUNG 1: AUFBAU UND TEST EINES SERVOS

In dieser Übung wird ein Servo mit einer Spannungsquelle und der BASIC Stamp verbunden. Wir werden uns davon überzeugen, dass sich die Position eines Servos über programmierte Signalsequenzen von der BASIC Stamp einfach steuern lässt.

Standard Servo

Abbildung 4-1 zeigt einen Parallax Standard Servo. Anschlussbuchse (1) verbindet den Servo mit einer Spannungsquelle (Vdd und Vss) und einer Signalquelle (ein BASIC Stamp I/O Pin). Das Kabel (2) übertragt Vdd, Vss und die Signale von der Anschlussbuchse zum Servo. Das Steuerhorn (3) ist Teil des Servos. Über die BASIC

Stamp wird das Steuerhorn angesteuert und führt kontrollierte Drehbewegungen aus. Im Gehäuse (4) sind Steuerschaltungen, der Gleichstrommotor und Getriebe untergebracht. Diese Bauteilgruppen arbeiten zusammen, empfangen die HIGH/LOW Signale der BASIC Stamp und wandeln sie in kontrollierte Drehbewegung des Steuerhorns um.

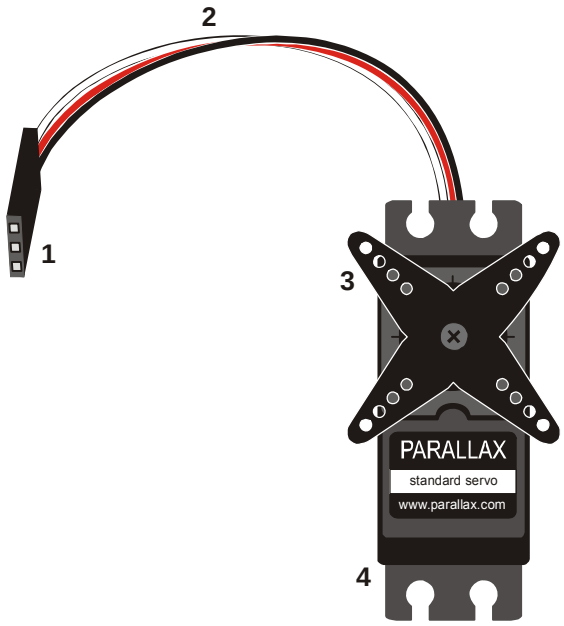


Abbildung 4-1
Der Parallax Standard Servo

(1) Buchse
(2) Kabel
(3) Steuerhorn
(4) Gehäuse

ACHTUNG: benutze für die Übungen in diesem Buch nur den Parallax Standard Servo!

Tausche den Standardservo von Parallax auf keinen Fall gegen einen anderen aus; benutze in den folgenden Versuchsschaltungen auf keinen Fall den Freilaufservo von Parallax, er könnte durch die in diesem Kapitel vorgestellten Schaltungen beschädigt werden.

Bauteile der Schaltung: Servo mit LED

Mit Hilfe einer LED Schaltung werden wir das Steuersignal, das die BASIC Stamp an den Servo schickt, überwachen. Diese Schaltung soll uns helfen, zusätzliche

Informationen über seine Arbeitsweise zu erhalten, für seine Ansteuerung ist sie nicht notwendig.

1 Parallax Standard Servo
1 Widerstand – 470 Ω (gelb-violett-braun)
1 LED – beliebige Farbe

Aufbau der Schaltung: Servo mit LED

Wenn du den Servo mit der BASIC Stamp verbindest, gehe besonders sorgfältig vor. Der Verbindungsaufbau hängt davon ab, ob das Board of Education (Rev B oder Rev C) oder das HomeWork Board benutzt. Auf dem Board of Education, befindet sich ein Hinweis zur Version (Rev B oder C) unterhalb der X4, X5 Wannenstecker (siehe Abbildung 4-2).

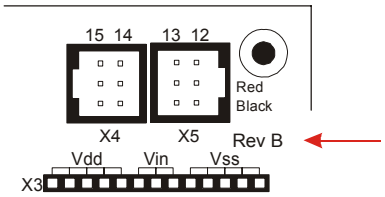


Abbildung 4-2
Board of Education Rev Hinweis

√ Prüfe nach, ob ein HomeWork Board oder ein Board of Education benutzt wird und bestimme beim BoE die Ausführungsversion Rev B oder C.

√ Die entsprechende Anleitung für die Verbindung des Servos mit der BASIC Stamp wird für die Boards auf den unten angegebenen Seiten gezeigt:

- Seite 111 – Board of Education Rev C
- Seite 114 – BASIC Stamp HomeWork Board
- Seite 117 – Board of Education Rev B

Board of Education Rev C

Abbildung 4-3 zeigt den Schaltungsaufbau für das Board of Education Rev C.

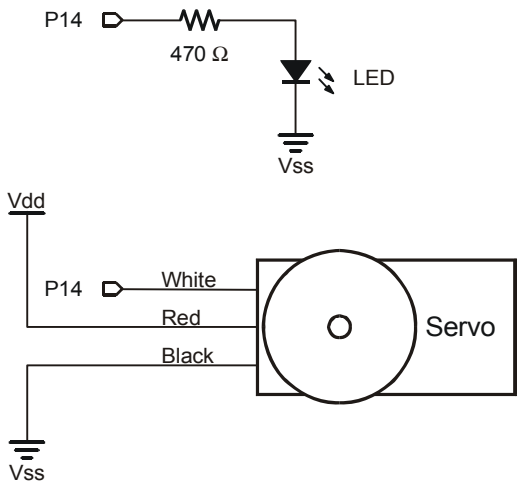


Abbildung 4-3
Servo und LED Kontrollleuchte Schaltungsaufbau für das Board of Education Rev C

√ Schalte die Spannungsversorgung ab (siehe Abbildung 4-4).

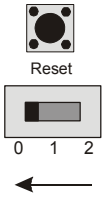


Abbildung 4-4
Spannungsversorgung Abschalten

Abbildung 4-5 zeigt die Servo Wannenstecker auf dem Board of Education Rev C. Über diese wird ein Servo angeschlossen. Auf dem Board befindet sich zwischen den beiden Wannensteckern ein Jumper, mit dem die Spannungsversorgung des Servos auf Vin oder Vdd gesetzt werden kann.

√ Setze den Jumper auf Vdd wie in Abbildung 4-5 gezeigt.

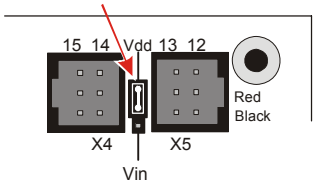


Abbildung 4-5
Servo Wannenstecker Jumper auf Vdd

Mit dem Jumper wird die Spannungsversorgung (Vin oder Vdd) des Parallax Standard Servo eingestellt. Wird mit einer 9 V Batterie gearbeitet, sollte der Jumper auf Vdd gesetzt werden. Bei einem 6 V Batterieblock kann jede der beiden Jumperstellungen benutzt werden. **Benutze Vdd nur** wenn du mit einem Gleichspannungsnetzteil arbeitest. Verwende nur Gleichspannungsquellen, die den Anforderungen im Anhang D: Batterien und genügen.

√ Baue die Schaltung nach Abbildung 4-6 auf.

√ Überprüfe, ob die Stecker richtig verbunden sind. Achte auf die Farbmarkierungen: von oben nach unten erscheinen weiß, rot und schwarz, wie in der Abbildung gezeigt.

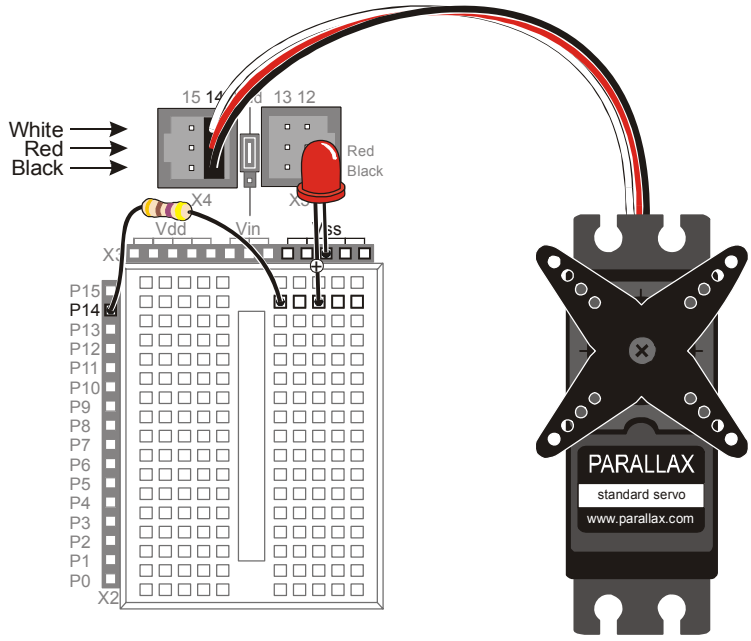


Abbildung 4-6
Servo und LED Kontrollleuchte des Board of Education Rev C

Bis jetzt haben wir den 3-fach Schiebeschalter nur in der Position 0 und 1 benutzt. Jetzt werden wir auch die Position 2 benötigen, in der Spannung an die Pins der Wannenbuchsen für den Servo gelegt wird.

√ Wähle die Stellung 2 des 3-fach Schiebeschalters (Abbildung 4-7); damit werden die Wannenstecker für den Servo mit der Spannungsquelle verbunden. Der Servo könnte sich ein wenig bewegen, sobald Spannung anliegt.

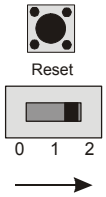


Abbildung 4-7
Einschalten der Spannungsversorgung für das Board of Education und die Servo Wannenstecker

√ Weiter geht es auf Seite 119: Programmierung einer Servosteuerung.

BASIC Stamp HomeWork Board

Nur wenn mit einem HomeWork Board gearbeitet wird, sollte dieser Abschnitt durchgelesen werden:

1 3-pin Stiftleiste (siehe Abbildung 4-8).
4 Verbindungsdrähte

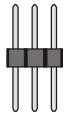


Abbildung 4-8
HomeWork Board oder Board of Education

3-Pin Stfitleiste

Abbildung 4-9 zeigt die LED- und Servo-Schaltungen auf dem HomeWork Board. In der nachfolgenden Beschreibung werden Hinweise für einen fehlerfreien Aufbau gegeben.

WARNUNG

Benutze für die Arbeit mit dem Standard Servo und dem BASIC Stamp HomeWork Board ausschließlich 9 V Batterien. Setze keine Steckernetzteile oder andere Gleichspannungsquellen ein. Der falsche Gebrauch dieser Geräte kann dazu führen, dass die Übungen nicht richtig funktionieren oder das der Servo beschädigt wird.

Arbeite möglichst mit neuen Batterien. Werden wiederaufladbare Batterien (Akkus) benutzt, sollten sie über eine Leistung von mindestens 100 mAh verfügen und vorher aufgeladen worden sein. Siehe auch im Anhang D: Batterien und .

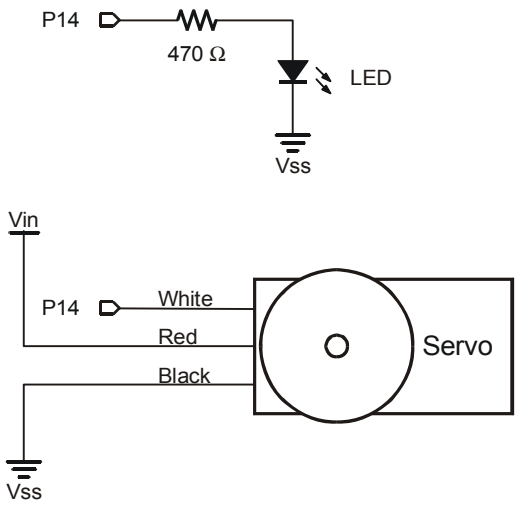


Abbildung 4-9
Schaltung für Servo und LED Anzeige auf dem HomeWork Board

√ Trenne die 9 V Batterie vom HomeWork Board.
√ Baue die Schaltungen LED Anzeige mit Servo-Stiftleiste, wie in Abbildung 4-10 gezeigt, auf.

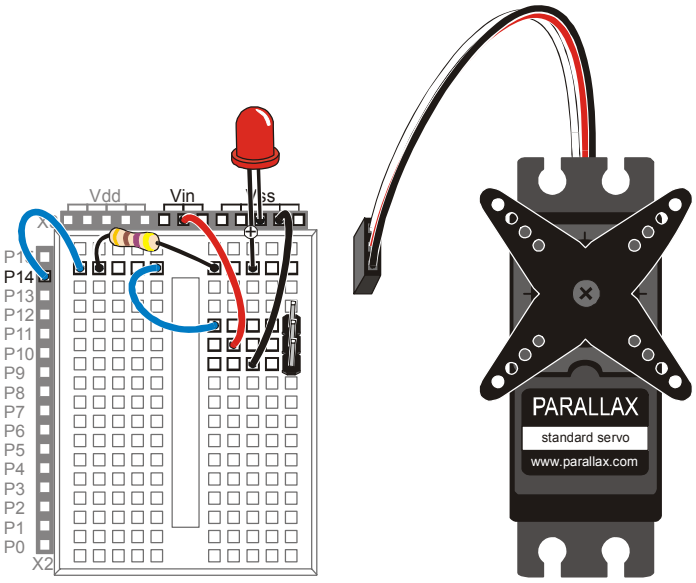


Abbildung 4-10 Schaltung LED Anzeige mit Servo-Stiftleiste auf dem HomeWork Board

√ Verbinde den Servo mit der Servo-Stiftleiste (Abbildung 4-11).
√ Achte auf die identische Farbreihenfolge der Drähte mit der Abbildung.
√ Überprüfe die Verdrahtung noch einmal.
√ Setze die 9 V Batterie wieder ein. Der Servo kann dabei eine kurze Bewegung ausführen.

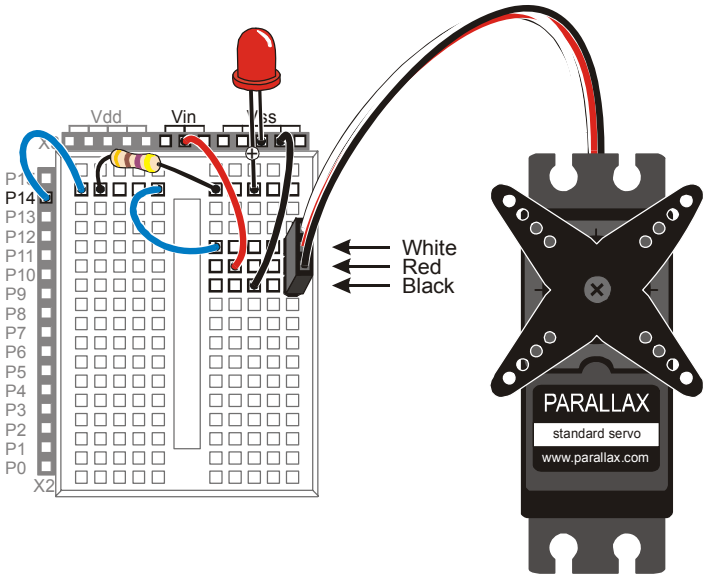


Abbildung 4-11
Verbinde den Servo mit der Stiftleiste auf dem HomeWork Board

√ Mache weiter auf Seite 119: **Programmierung einer Servosteuerung**.

Board of Education Rev B

Abbildung 4-12 zeigt die Schaltung für den Servo und die LED Schaltung auf einem Board of Education Rev B. In der nachfolgenden Beschreibung werden Hinweise für einen fehlerfreien Aufbau gegeben.

WARNUNG

Benutze für die Arbeit mit dem Standard Servo und einem Board of Education Rev B ausschließlich 9 V Batterien. Setze keine Steckernetzteile oder andere Gleichspannungsquellen ein. Der falsche Gebrauch dieser Geräte kann dazu führen, dass die Übungen nicht richtig funktionieren oder das der Servo beschädigt wird.

Arbeite möglichst mit neuen Batterien. Werden wiederaufladbare Batterien (Akkus) benutzt, sollten sie über eine Leistung von mindestens 100 mAh verfügen und vorher aufgeladen worden sein. Siehe auch im Anhang D: Batterien und .

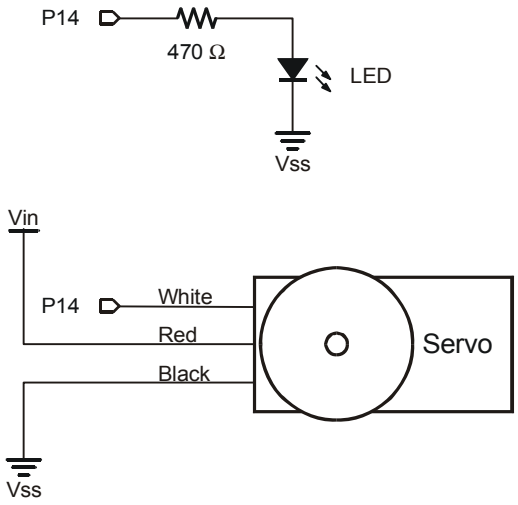


Abbildung 4-12
Schaltung für Servo und LED Anzeige auf einem Board of Education Rev B

√ Trenne die Batterie oder jede andere Energiequelle vom Board.
√ Baue die LED Schaltung aus Abbildung 4-12 auf.
√ Verbinde den Servo mit der Stiftleiste wie in Abbildung 4-13 gezeigt.

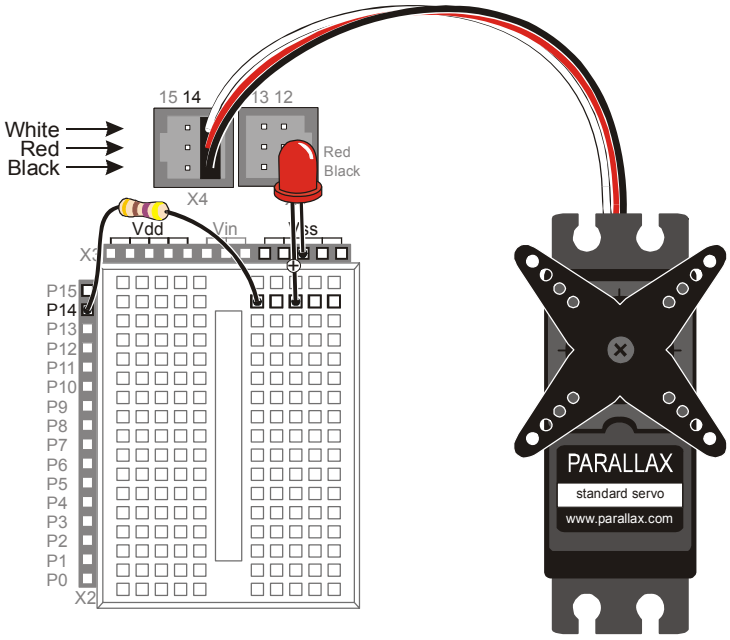


Abbildung 4-13
Verbinde den Servo mit der Stiftleiste auf dem Board of Education Rev B.

√ Achte darauf, das die Reihenfolge der farbigen Drähte mit der Darstellung in der Abbildung übereinstimmt.

√ Schließe eine 9 V Batterie an das Board of Education Rev B. Der Servo kann sich bei Herstellung der Verbindung etwas bewegen.

Programmierung einer Servosteuerung

Servos werden über Serien kurzer Impulse zwischen 1ms und 2 ms gesteuert; die Impulse werden alle 20 ms gesendet. Der Befehl **`PULSOUT`** erzeugt einen Impuls (ein sehr kurzes Signal) an einem BASIC Stamp I/O Pin. Die Befehlssyntax für einen **`PULSOUT`** Befehl lautet:

PULSOUT *Pin, Duration*

Wie wir schon bei den Befehlen **`HIGH`** und **`LOW`** gesehen haben, ist das Argument von ***`Pin`*** eine Zahl, die der BASIC Stamp mitteilt, an welchen I/O Pin das Signal zu senden ist. Das Argument ***`Duration`*** wird hier nicht in der Zeiteinheit ms angegeben, sondern ist der 2-millionste Teil einer Sekunde (µs) (gilt nur für eine BASIC Stamp 2). Die Zeit legt die Länge eines Impulses (in 0,5 µs) fest.

Den millionsten Teil einer Sekunde bezeichnet man auch als Mikrosekunde. Als Abkürzung für die Vorsilbe Mikro wird der griechische Buchstabe µ genommen, s steht für Sekunde. 2 Mikrosekunden lassen sich so kürzer mit 2 µs angeben.

Erinnerung: ein tausendstel einer Sekunde bezeichnen wir als Millisekunde oder kürzer: ms.

Umrechnung: 1 ms = 1000 µs. Mit anderen Worten, es passen in eine tausendstel Sekunde 1000 Zeitanteile von je 1-millionstel Sekunde.

Im folgenden Programmbeispiel werden wir mit dem Befehl **`PULSOUT`** Impulse an den Servo übertragen und ihn damit anweisen, das Steuerhorn in eine bestimmte Position zu drehen. Mit Hilfe von **`FOR…NEXT`** Schleifen werden wir eine feste Anzahl von Impulsen an den Servo senden und ihn für eine bestimmte Zeit in einer Position halten.

Programmbeispiel: ServoTest.bs2

Mit dem Programm ServoTest.bs2 versuchen wir das Steuerhorn für ca. drei Sekunden in der Position 10 Uhr zu halten. Anschließend lassen wir das Steuerhorn für ca. drei Sekunden im Uhrzeigersinn bis in die 2 Uhr Position drehen und abschließend fixieren wir es für drei Sekunden in der 12 Uhr Position .

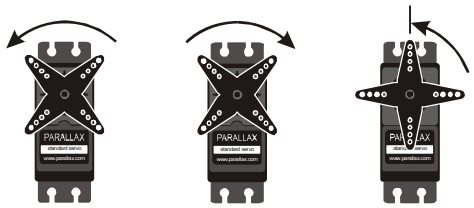


Abbildung 4-14
Steuerhornbewegung des Servo

10 Uhr (links)
2 Uhr (Mitte)
12 Uhr (rechts)

Was mache ich, wenn ich einen anderen Servo habe? Wir empfehlen für diese Übungen den Parallax Standard Servo. Es gibt viele unterschiedliche Servos, die anders als beschrieben, auf das Programm TestServo.bs2 reagieren könnten. So lassen sich einige Servos zum Beispiel nur in feste Positionen drehen. Sollte dein Servo für 9 V ausgelegt sein und reagiert er ähnlich wie in dieser Übung beschrieben, dann wird er auch in den anderen Übungen einsetzbar sein. Du kannst natürlich auch die Programmbeispiele ändern und damit den Servo nach eigenen Vorstellungen ansteuern.

√ Gib das Programm ServoTest.bs2 in den BASIC Stamp Editor ein.

- √ Schließe das Board of Education oder HomeWork Board an eine Spannungsquelle an.
- √ Führe das Programm aus.
- √ Beobachte, ob sich der Servo bei jedem der drei Programmblöcke, so wie beschrieben, bewegt und das Steuerhorn in die entsprechende Richtung zeigt.
- √ Starte das Programm erneut und überzeuge dich, dass die LED leicht flackert. Sie sollte am hellsten aufleuchten, wenn die BASIC Stamp das 10-Uhr Signal sendet und am schwächsten beim 2-Uhr-Signal. Das liegt daran, dass die LED Schaltung nur halb so lange eingeschaltet wird. (1 von 21 ms statt 2 von 22 ms).

```
' What's a Microcontroller - ServoTest.bs2
' Test the servo at three different position signals.

' {$STAMP BS2}
' {$PBASIC 2.5}

counter        VAR     Word

DEBUG "Counterclockwise 10 o'clock", CR

FOR counter = 1 TO 150
  PULSOUT 14, 1000
  PAUSE 20
NEXT

DEBUG "Clockwise 2 o'clock", CR

FOR counter = 1 TO 150
  PULSOUT 14, 500
  PAUSE 20
NEXT

DEBUG "Center 12 o'clock", CR

FOR counter = 1 TO 150
  PULSOUT 14, 750
  PAUSE 20
NEXT

DEBUG "All done."

END
```

Arbeitsweise von ServoTest.bs2

Die erste **FOR…NEXT** Schleife liefert 150 Impulse, von je 2 ms Dauer. Diese Impulse weisen den Servo an, das Steuerhorn in die Position von ca. 10.00 Uhr zu bewegen.

```
FOR counter = 1 TO 150
  PULSOUT 14, 1000
  PAUSE 20
NEXT
```

PULSOUT 14,1000 sendet einen Impuls von 1000 × 2 µs. Das sind 2000 µs oder 2 ms.

Abbildung 4-15 ist ein Zeitdiagramm, in dem die HIGH und LOW Zustände eines Signals mit seinen Längen wiedergegeben sind. In ihm wird nicht gezeigt, wie viele Impulse ausgesandt werden, sondern nur wie lang die HIGH und LOW Zustände dauern. Jeder Puls (HIGH) dauert 2.0 ms, ihm folgt ein 20 ms LOW Signal.

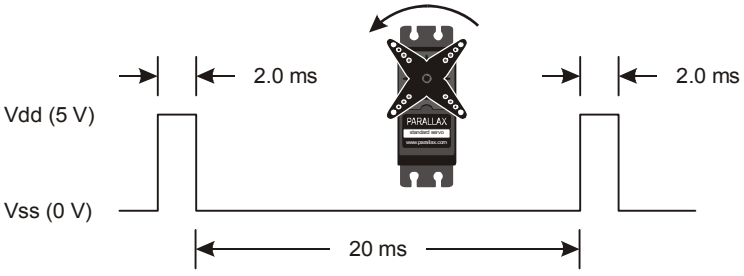


Abbildung 4-15
Zeitdiagramm für Signal HIGH von 2.0 ms Länge gefolgt von einem LOW Signal von 20 ms.

Steuerhorn in Position 10.00 Uhr.

Die zweite **FOR…NEXT** Schleife liefert 150 Impulse, mit einer Länge von je 1.0 ms. Damit wird der Servo für ca. 3.15 Sekunden in der 14.00 Uhr Position gehalten.

```
FOR COUNTER = 1 TO 150
  PULSOUT 14, 500
  PAUSE 20
NEXT
```

PULSOUT 14,500 sendet einen Impuls von 500 × 2 µs Länge. Das entspricht 1000 µs oder 1 ms.

Abbildung 4-15 zeigt das Zeitdiagramm. Die Pausen (Signal LOW) zwischen zwei Impulsen (Signal HIGH) dauern 20 ms.

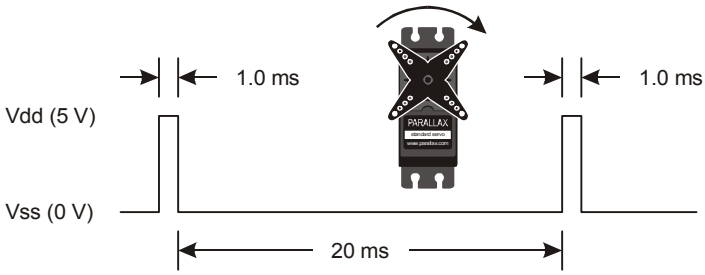


Abbildung 4-16
Zeitdiagramm für Impulsfolge aus 1.0 ms HIGH und 20 ms LOW

Steuerhorn in Position 14.00 Uhr.

Die letzte **FOR…NEXT** Schleife liefert 150 Impulse der Länge 1.5 ms. Damit wird der Servo für ca. 3,23 Sekunden in der Mittelposition (12.00 Uhr) gehalten.

```
FOR counter = 1 TO 150
  PULSOUT 14, 750
  PAUSE 20
NEXT
```

PULSOUT 14, 750 sendet einen Impuls von 750 × 2 µs. Das sind 1500 µs oder 1.5 ms.

Abbildung 4-17 zeigt das Zeitdiagramm dieser Impulsfolge. Der Zustand Signal HIGH dauert 1,5 ms, Signal LOW 20 ms.

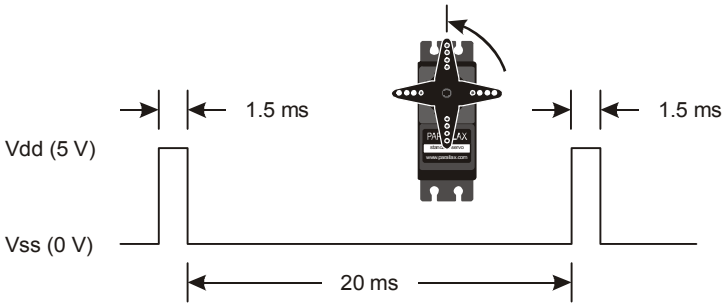


Abbildung 4-17
Zeitdiagramm für Signal HIGH von 1.5 ms Länge und Signal Low von 20 ms

Steuerhorn in Position 12.00 Uhr.

Etwas Mathematik

Berechnung *Duration* aus der Impulslänge: Verwende für die Umrechnung von Millisekunden in ein Zeitformat, dass im Befehl **PULSOUT** benötigt wird, die Gleichung:

$$Duration = number\ of\ ms \times 500$$

Beispiel: Um für eine Impulslänge von 1,5 ms das Argument des Befehls **PULSOUT** zu bestimmen, setze den Zahlenwert von 1,5 in die obige Gleichung ein und berechne den Wert.

$$Duration = 1.5 \times 500$$
$$= 750$$

Berechnung *Impulslänge (in ms)* aus Duration: Die Impulslänge eines Signals eines beliebigen **PULSOUT** Befehls ergibt sich über die Formel:

$$number\ of\ ms = \frac{Duration}{500} ms$$

Beispiel: Ist der Befehl **PULSOUT 14, 850** gegeben, wie groß ist dann die Impulslänge?

$$number\ of\ ms = \frac{850}{500} ms$$
$$= 1.7\ ms$$

Du bist dran – Positionierung und Haltezeit

Die Zahl der Schleifenwiederholungen bei einer **FOR…NEXT** Schleife bestimmt, wie lange ein Servo in einer bestimmten Position bleibt. Der Wert des Arguments ***Duration*** im Befehl **PULSOUT** bestimmt die Drehrichtung. Experimentiere mit diesem Befehl, indem du die Werte veränderst, dir vorher Klarheit verschaffst, in welche Richtung der Servo dreht und dann das Realexperiment durchführst und deine Vorhersage überprüfst. Gehe erst dann zum nächsten Experiment über, wenn du Sicherheit im Umgang mit dem Steuerbefehl für den Servo erlangt hast.

- √ Speicher ServoTest.bs2 unter ServoTestYourTurn.bs2.
- √ Ändere alle **FOR...NEXT** Schleifen so ab, dass nur halb so viele Durchläufe wie im Originalprogramm ausgeführt werden:

```
FOR counter = 1 to 75
```

- √ Führe das abgeänderte Programm aus und überprüfe, ob der Servo jede Position nur halb so lange beibehält.
- √ Änder alle **FOR...NEXT** Schleifen so ab, dass doppelt so viele Durchläufe wie im Originalprogramm ausgeführt werden:

```
FOR counter = 1 to 300
```

√ Führe das abgeänderte Programm aus und überprüfe, ob der Servo jede Position doppelt so lange beibehält.

√ Ändere den Befehl **PULSOUT** in der ersten Schleife wie folgt ab:

```
PULSOUT 14,850
```

√ Ändere den Befehl **PULSOUT** in der zweiten Schleife wie folgt ab:

```
PULSOUT 14,650
```

√ Führe das geänderte Programm aus und erkläre die unterschiedlichen Positionen und Haltezeiten des Servos.

ÜBUNG 2: POSITIONSSTEUERUNG MIT DEM COMPUTER

Industrielle Automatisierung basiert häufig auf dem Einsatz von Mikrocontrollern, die mit großen Rechnern verbunden sind und über Sensoren aufgenommenen Daten dorthin übertragen. Der Zentralrechner wertet die Daten aus und gibt Steuerdaten an den Mikrocontroller zurück. Auf diese Weise lassen sich zum Beispiel bei einem Formel-1 Rennen alle wichtigen Motordaten eines Rennbolliden an den Zentralrechner per Funk übertragen und dieser gibt entsprechende Hinweise und Verhaltensweisen an die Techniker und den Fahrer zurück

Über das Debug Terminal kannst du Informationen oder Hinweise vom Computer an die BASIC Stamp übertragen (Abbildung 4-18). Die BASIC Stamp muss dafür so programmiert werden, dass sie die Mitteilungen, die vom Debug Terminal verschickt werden, auch versteht; die Daten müssen in Variablen abgelegt werden.

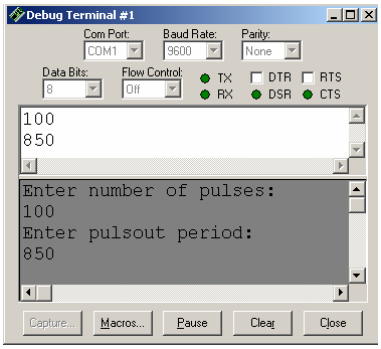


Abbildung 4-18
Mitteilungen an die BASIC Stamp übertragen

Klicke auf das weiße Feld oberhalb des Mitteilungsfensters und gib eine Mitteilung ein. Eine Kopie der Mitteilung erscheint im unteren Fenster. Diese Kopie bezeichnen wir als Echo.

In dieser Übung wird die BASIC Stamp so programmiert, dass sie zwei Werte über das Debug Terminal einliest:

1. die Zahl der Impulse, die an den Servo übertragen werden.
2. den ***Duration*** Wert, der vom Befehl **PULSOUT** benötigt wird.

Mit Hilfe dieser beiden Werte soll die BASIC Stamp so programmiert werden, dass sie den Servo steuert.

Bauteile und Schaltung

Wie in Übung 1

BASIC Stamp empfängt Steuerinformationen von DEBUG

Mit Hilfe des **DEBUG** Befehls sendet die BASIC Stamp Daten/Mitteilungen an das Debug Terminal. Mit dem Befehl **DEBUGIN** empfängt die BASIC Stamp Daten/Mitteilungen vom Debug Terminal. Vorher müssen eine oder mehrere Variable deklariert werden, in die die erhaltenen Werte/Informationen abgelegt werden können. Das folgende Beispiel zeigt die Deklaration einer Variablen:

```
pulses VAR Word
```

Später kann dieser Variablen über das Programm ein Wert zugewiesen werden, der über den Befehl **DEBUGIN** eingelesen wird:

```
DEBUGIN DEC pulses
```

Empfängt die BASIC Stamp einen nummerischen Wert vom Debug Terminal, wird er in der Variablen **`pulses`** abgelegt. Die Formatierungsanweisung **`DEC`** teilt dem Befehl **`DEBUGIN`** mit, das die gesendeten Zeichen ganze Zahlen sind, die insgesamt eine Dezimalzahl darstellen. Sobald die Rücklauftaste gedrückt wird, legt die BASIC Stamp die bis dahin empfangenen Zahlen in der Variablen **`pulses`** als Dezimalzahl ab und fährt im Programm weiter fort.

Die folgende Zeile ist nicht im Programmbeispiel enthalten, kann aber eingefügt werden, um anzuzeigen, dass die BASIC Stamp den Befehl ausgeführt hat.

```
DEBUG CR, "You sent the value: ", DEC pulses
```

Programmbeispiel: ServoControlWithDebug.bs2

Abbildung 4-19 zeigt eine Ausschnittsvergrößerung des Fensters *Übertragen* im *Debug Terminal*. Der Ausschnitt kann im aktuellen Fall kleiner ausfallen. Mit einem Klick und Ziehen auf den Separator zwischen den beiden Fenstern lässt sich das *Übertragen*-Fenster vergrößern. In ihm werden Zeichen eingeben, die an die BASIC Stamp übertragen werden. In dem dargestellten Beispiel wurde 100 eingegeben, die Zeilenrücklauftaste gedrückt und eine weitere Zahl 850 eingetippt. Im Fenster *Empfangen* erscheint die von der BASIC Stamp abgeschickte Mitteilung “Enter number of pulses:” . Ebenso wird das Echo der Dateneingabe (100) angezeigt.

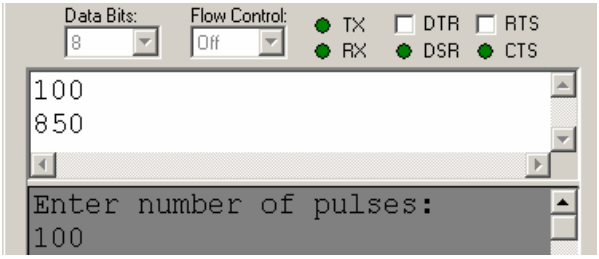
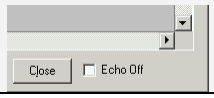


Abbildung 4-19
Fenster im Debug Terminal:

← *Fenster Senden*

← *Fenster Empfangen*

Echo bezeichnet eine Mitteilung, die, einmal gesendet, als Kopie im Fenster *Empfangen* erscheint. Aktiviere das Markierungsfeld *Echo Off* (siehe unten), wenn Echos unterdrückt werden sollen.



- √ Gib das Programm ServoControlWithDebug.bs2 in den BASIC Stamp Editor ein und führe es aus.
- √ Sollte das Fenster *Senden* zu klein sein, kann es vergrößert werden, wenn man den Separator bei gedrückter Maustaste verschiebt. Der Separator befindet sich direkt über der Textzeile: "Enter number of pulses:" (siehe Abbildung 4-19).
- √ Klicke in das Fenster *Senden* und gib an der Cursorposition eine Mitteilung ein.
- √ Wenn das Debug Terminal mit der Mitteilung "Enter number of pulses:" zur Dateneingabe auffordert, gib die Zahl 100 ein und drücke die Zeilenrücklauftaste.
- √ Erscheint die Eingabeaufforderung "Enter PULSOUT duration:" auf dem Debug Terminal, gib die Zahl 850 ein und drücke die Taste Enter.

Als Argument *Duration* im Befehl PULSOUT sollte eine Zahl zwischen 500 und 1000 gewählt werden. Werden Zahlen außerhalb des angegebenen Zahlenbereiches eingegeben, könnte der Servo in Positionen drehen, die seine mechanischen Grenzen überschreiten und damit seine Lebensdauer deutlich verkürzen.

Die BASIC Stamp zeigt jetzt an "Servo is running…" während die Impulse an den Servo übertragen werden. Nach erfolgter Übertragung erscheint eine Sekunde lang die Mitteilung: "Done" bevor das Programm wieder zur Eingabe der Impulszahl ("Enter number of pulses:") auffordert. Achte bei den Versuchen unbedingt darauf, dass der *Duration*-Wert des Befehls **PULSOUT** zwischen 500 und 1000 liegt.

- √ Experimentiere bei **PULSOUT *Duration*** mit Werten zwischen 500 und 1000 und variiere die Werte für *pulses* zwischen 1 und 65534.

Es werden zwischen 40 und 45 Impulse benötigt, um einen Servo 1 Sekunde lang in einer Position zu halten.

```
' What's a Microcontroller - ServoControlWithDebug.bs2
' Send messages to the BASIC Stamp to control a servo using
' the Debug Terminal.

' {$STAMP BS2}
' {$PBASIC 2.5}

counter        Var     Word
pulses         Var     Word
duration       Var     Word
```

```
DO

  DEBUG CLS, "Enter number of pulses:", CR
  DEBUGIN DEC pulses

  DEBUG "Enter PULSOUT duration:", CR
  DEBUGIN DEC duration

  DEBUG "Servo is running...", CR

  FOR counter = 1 TO pulses
    PULSOUT 14, duration
    PAUSE 20
  NEXT

  DEBUG "DONE"
  PAUSE 1000

LOOP
```


Arbeitsweise von ServoControlWithDebug.bs2

Am Anfang des Programms werden drei Variable vom Typ **Word** deklariert:

```
counter         Var     WORD
pulses          Var     WORD
duration        Var     WORD
```

Die Variable **counter** wird innerhalb einer **FOR…NEXT** Schleife benötigt. Weitere Einzelheiten finden sich in Kapitel2, Übung 3. Die Variablen **pulses** und **duration** werden zu verschiedenen Zwecken benötigt. Beide senden und empfangen Werte vom Debug Terminal. Die Variable **pulses** bestimmt in der **FOR…NEXT** Schleife, mit der Impulse an den Servo übertragen werden, auch die Anzahl der Wiederholungen und die **duration** Variable legt die Impulslänge im Befehl **PULSOUT** fest.
Der Rest des Programms ist in eine **DO…LOOP** Endlosschleife ohne Abbruchbedingung (mit **WHILE** oder **UNTIL**) eingebettet.

```
DO
  ' Rest of program not shown.
LOOP
```

Mit Hilfe des Befehls **DEBUG** wird der Anwender über eine Mitteilung aufgefordert, die Anzahl der Impulse einzugeben. Der Befehl **DEBUGIN** wartet nun so lange, bis der Anwender eine Zahl eingegeben und die Zeilenrücklauftaste (Enter) gedrückt hat. Die

eingegebene Zahl wird in der Variablen **pulses** abgelegt und in einem zweiten Aufruf wird wiederum mit den Befehlen **DEBUG** und **DEBUGIN** der Wert für die Variable **duration** eingelesen.

```
DEBUG CLS, "Enter number of pulses:", CR
DEBUGIN DEC pulses

DEBUG "Enter PULSOUT duration:", CR
DEBUGIN DEC duration
```

Nachdem beide Werte eingegeben worden sind, wird der Anwender durch eine Mitteilung darauf hingewiesen, das der Servo angesteuert wird:

```
DEBUG "Servo is running...", CR
```

Während der Servo in seine Position dreht, versuche einmal die Bewegung mit der Hand aufzuhalten. Du wirst feststellen, dass das Steuerhorn eine gewisse Kraft ausübt.

FOR Counter = StartValue TO EndValue {STEP StepValue}…NEXT

Dies ist die Syntax einer **FOR…NEXT** Schleife, wie im BASIC Stamp Manual dargestellt. ***Counter***, ***StartValue*** und ***EndValue*** legen fest, wie häufig die Schleife durchlaufen wird. Der optionale ***StepValue*** Wert bestimmt, um welchen Wert ***Counter*** bei jedem Schleifendurchlauf erhöht wird. ***Counter*** wird auch als Kontrollvariable bezeichnet. Ihr Wert wird bei jedem Schleifendurchlauf um 1 erhöht, solange ***StepValue*** nicht aufgerufen wird.

In allen vorangegangenen Beispielen wurde die Kontrollvariabale **counter** mit einem vorher festgelegten Wert in einer **FOR…NEXT** Schleife benutzt. Als Endwert wurden für **EndValue** bis jetzt nur die Zahlen 10 oder 150 eingegeben. In der folgenden **FOR…NEXT** Schleife steuert der Wert der Variablen *pulses* den Wert von **EndValue.** Auf diese Weise wird überwacht, wie viele Impulse an den Servo gesendet werden. Die Variable **pulses** legt fest, wie lange ein Servo in einer bestimmten Position verharrt. Als Argumentwert ***Duration*** im Befehl **PULSOUT** wurden bisher konstante Werte der Größe 500, 750 und 1000 benutzt. Versuche die Arbeitsweise der folgenden **FOR…NEXT** Schleife genau zu verstehen und schaue dir an, wie die Variablen aufgerufen werden:

```
FOR counter = 1 to pulses
  PULSOUT 14, duration
  PAUSE 20
NEXT
```

Studiere die `FOR…NEXT` Schleife genau und achte einmal darauf, wie die Variablen eingesetzt werden.

Du bist dran – Grenzen setzen

Im vorherigen Programmbeispiel kann ein Anwender als Wert für **`PULSOUT`** ***`duration`*** zum Beispiel 1500 eingeben und damit dem Servo schaden. Das Programm sollte jetzt so abgeändert werden, dass nicht zugelassene Zahleneingaben (aus Sicht des Entwicklers) auch nicht angenommen werden.

Stellen wir uns vor, dass eine computerbasierte Steuerung des Servos entwickelt wurde, die den Schließ- und Öffnungsmechanismus einer Tür fernsteuert. Es könnte sich zum Beispiel um ein Luk auf einem Transportschiff handeln, das von einem Sicherheitsingenieur über ein Kamerasystem überwacht wird. Oder es handelt sich um ein wissenschaftliches Experiment, bei dem ein Student durch das gezielte Schließen und Öffnen von Gattern innerhalb eines Labyrinths, das Verhalten von Mäusen bei der Nahrungssuche studiert und dokumentiert. Wenn also ein Produkt entworfen wird, dass von anderen Menschen bedient werden soll, muss man versuchen es so zu gestalten, dass Fehlbedienungen (nahezu) ausgeschlossen sind. Eingabefehler durch den Anwender müssen deshalb über die Software nach Möglichkeit abgefangen werden.

Das könnten so aussehen:

- √ Speicher das Beispielprogramm ServoControlWithDebug.bs2 unter dem neuen Namen ServoControlWithDebugYourTurn.bs2.
- √ Ersetze die beiden Befehlszeilen:

```
DEBUG "Enter pulsout duration:", CR
DEBUGIN DEC duration
```

durch den Programmblock:

```
DO
  DEBUG "Enter pulsout duration:", CR
  DEBUGIN DEC duration
  IF duration < 500 THEN
    DEBUG "Value of duration must be above 499", CR
    PAUSE 1000
  ENDIF
  IF duration > 1000 THEN
    DEBUG "Value of duration must be below 1001", CR
    PAUSE 1000
  ENDIF
```

```
  LOOP UNTIL duration > 499 AND duration < 1001
```

√ Speicher das Programm.
√ Führe das Programm aus und überzeuge dich, dass Werte, die außerhalb des Wertebereiches eines Servos liegen, vom Programm nicht angenommen werden.

ÜBUNG 3: STEUERUNG VON BEWEGUNG

In dieser Übung werden wir ein Programm schreiben, mit dem ein Servo Positionsänderungen unterschiedlich schnell vornimmt, das Steuerhorn wird dabei unterschiedlich schnell, d.h. mit unterschiedlicher Geschwindigkeit, in eine neue Position gedreht. Mit dieser Technik beeinflussen wir über einen Servo Bewegungen.

Programmierung unterschiedlich schneller Drehbewegungen

Mit Hilfe der folgenden **FOR…NEXT** Schleife wird der Stellbereich eines Servo durchfahren:

```
FOR counter = 500 TO 1000
  PULSOUT 14, counter
  PAUSE 20
NEXT
```

Die **FOR…NEXT** Schleife lässt das Steuerhorn aus der 14.00 Uhr Position starten und bewegt es dann langsam entgegen dem Uhrzeigersinn in die 10.00 Uhr Position. Die Kontrollvariable **counter** wird in der **FOR…NEXT** Schleife bei jedem Durchlauf jeweils um 1 erhöht. Der Wert der Kontrollvariablen wird auch über das Argument ***Duration*** im Befehl **PULSOUT** abgerufen; d.h. dass die Pulsdauer (***duration***) sich bei jedem Schleifendurchlauf etwas erhöht. Da sich ***duration*** ändert, ändert sich auch die Position des Steuerhorns.

Die Schrittweite in **FOR…NEXT** Schleifen wird optional durch das Argument **STEP** eingestellt. Mit Hilfe des **STEP** Wertes beeinflusst man die Drehgeschwindigkeit des Servos (je größer die Schrittweite, desto höher die Geschwindigkeit). Wird zum Beispiel das Argument von **STEP** auf 8 gesetzt - die Kontrollvariable **counter** erhöht bei jedem Durchlauf ihren Wert um 8 statt wie bisher um 1 - dann sieht die zugehörige **FOR** Anweisung wie folgt aus:

```
FOR counter = 500 TO 1000 STEP 8
```

Wird innerhalb eines Schleifendurchlaufs von einer hohen zu einer niedrigeren Zahl zurückgezählt, kehrt sich die Drehbewegung des Servo um. In PBASIC werden **FOR…NEXT** Schleifen rückwärtszählend durchlaufen, wenn der Startwert ***StartValue*** größer als der Endwert ***EndValue*** ist. Das folgende Beispiel zeigt eine Anwendung einer rückwärts zählenden **FOR…NEXT** Schleife:

```
FOR counter = 1000 TO 500
```


Optional kann die Schrittweite einer rückwärts zählenden Schleife mit **STEP** verändert und damit die Drehgeschwindigkeit des Servos erhöht werden:

```
FOR counter = 1000 TO 500 STEP 20
```

Halten wir fest: Einen Servo unterschiedlich schnell Drehbewegungen nach rechts oder links ausführen zu lassen, wird über die Angabe der Schrittweite innerhalb einer auf- oder abwärtszählenden **FOR…NEXT** Schleife gesteuert. Im nächsten Beispiel nutzen wir diese Techniken, um das Steuerhorn unterschiedlich schnell rechts- oder linksdrehend zu bewegen.

Programmbeispiel: ServoVelocities.bs2

- √ Gib das Programm ein und starte es.
- √ Beobachte während des Programmlaufs wie sich der Wert der Variablen **counter** im Debug Terminal verändert.
- √ Beobachte auch, wie unterschiedlich sich der Servo bei den beiden **FOR…NEXT** Schleifendurchläufen verhält. Drehrichtung und Geschwindigkeit des Servos werden verändert.

```
' What's a Microcontroller - ServoVelocities.bs2
' Rotate the servo counterclockwise slowly, then clockwise rapidly.

' {$STAMP BS2}
' {$PBASIC 2.5}

counter        VAR     Word

DO

DEBUG "Pulse width increment by 8", CR

  FOR counter = 500 TO 1000 STEP 8
    PULSOUT 14, counter
    PAUSE 7
    DEBUG DEC5 counter, CR, CRSRUP
  NEXT
```

```
  DEBUG CR, "Pulse width decrement by 20", CR

  FOR counter = 1000 TO 500 STEP 20
    PULSOUT 14, counter
    PAUSE 7
    DEBUG DEC5 counter, CR, CRSRUP
  NEXT

  DEBUG CR, "Repeat", CR

LOOP
```

Arbeitsweise von ServoVelocities.bs2

Die erste **FOR…NEXT** Schleife zählt aufsteigend von 500 bis 1000, mit einer Schrittweite von 8. Die Variable **counter** wird im Argument **Duration** von **PULSOUT** eingesetzt und dreht das Steuerhorn entgegen dem Uhrzeigersinn viermal schneller als im Normalfall (Step = 1).

```
FOR counter = 500 TO 1000 STEP 8
  PULSOUT 14, counter
  PAUSE 7
  DEBUG DEC5 counter, CR, CRSRUP
NEXT
```

Warum PAUSE 7 statt PAUSE 20? Die Ausführungszeit für den Befehl **DEBUG DEC5 counter, CR, CRSRUP** benötigt 8 ms. Das heißt, dass **PAUSE 12** eine Verzögerung von 20 ms zwischen den Pulsen aufrecht erhalten würde. Verschiedene Versuche haben gezeigt, dass die Bewegung des Servo bei **PAUSE 7** recht gleichmäßig ausgeführt wird. Dein Servo könnte anders reagieren.

Weitere DEBUG Formatanweisungen und Kontrollzeichen werden mit dem Befehl **DEBUG** bei der Ausgabe der Variablen **counter** vorgestellt. Der Wert von **counter** wird im Dezimalformat mit 5 Stellen ausgegeben (**DEC5**); anschließend erfolgt ein Zeilenvorschub (**CR**) und die Formatanweisung **CRSRUP** (cursor up) bewegt den Cursor in die linke obere Zeile des Bildschirms. Neue Werte überschreiben auf diese Weise ständig die vorherigen Werte.

Die zweite **FOR…NEXT** Schleife zählt rückwärts von 1000 nach 500 mit einer Schrittweite von 20. Die Variable **counter** wird in diesem Beispiel auch als Argument des Befehls **PULSOUT** eingesetzt, so dass der Servo sich in Uhrzeigerrichtung dreht.

```
FOR counter = 1000 TO 500 STEP 20
```

```
    PULSOUT 14, counter
    PAUSE 7
    DEBUG DEC5 counter, CR, CRSRUP
  NEXT
```

Du bist dran – Geschwindigkeiten anpassen

- √ Versuche mit unterschiedlichen Werten von **STEP** den Servo unterschiedlich schnell rotieren zu lassen.
- √ Starte das Programm nach jeder Programmänderung neu.
- √ Beobachte wie unterschiedlich schnell sich das Steuerhorn bei jeder neuen Einstellung von **STEP** dreht.
- √ Experimentiere mit unterschiedlichen ***Duration*** Werten im Befehl **PAUSE** (zwischen 3 und 12) und bestimme für jeden neuen **STEP** Wert, mit welchem Einstellwert der Servo die ruhigste Drehbewegung ausführt.

ÜBUNG 4: SERVOSTEUERUNG MIT TASTER

In diesem Kapitel hast du bereits Programme geschrieben, die einen Servo verschiedene Bewegungen ausführen lassen, dabei wurde die Steuerung über das Debug Terminal vorgenommen. Die Steuerung von Servos kann über die BASIC Stamp auch von Tastern übernommen werden. In dieser Übung wird:

- eine Schaltung für eine Servosteuerung mit Taster aufgebaut.
- die BASIC Stamp so programmiert, dass die Steuerung des Servos über die Eingabe des Tasters erfolgt.

Nach Fertigstellung wird sich der Servo, gesteuert über die BASIC Stamp, beim Drücken eines Tasters in die eine Richtung und beim Drücken eines zweiten Tasters in die andere Richtung drehen. Wird kein Taster gedrückt, bleibt der Servo in der vorhandenen Position stehen.

Zusatzbauteile für Servosteuerung mit Taster

Es werden die gleichen Bauteil wie in der vorangegangenen Übung benötigt. Zusätzlich erforderlich sind:

2 Taster - Schließer
2 Widerstände – 10 kΩ (braun-schwarz-orange)
2 Widerstände – 220 Ω (rot-rot-braun)

3 Verbindungsdrähte

Schaltung Tastersteuerung hinzufügen

Abbildung 4-20 zeigt die Taster-Schaltung, mit der der Servo angesteuert wird.

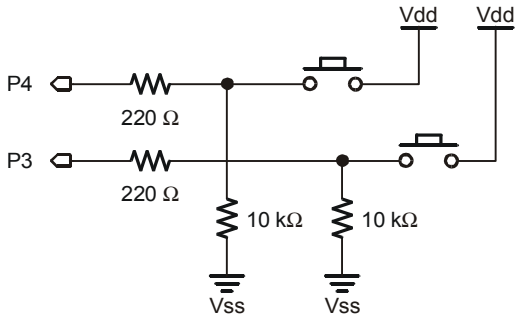


Abbildung 4-20
Schaltung Taster für Servo Ansteuerung

√ Füge zu der bis jetzt genutzten Servo+LED Schaltung die Tasterschaltung hinzu. Nach Fertigstellung sollte die Schaltung ähnlich aussehen wie in:

- Abbildung 4-21, wenn mit dem Board of Education Rev C gearbeitet wird.
- Abbildung 4-22, wenn das HomeWork Board benutzt wird.
- Abbildung 4-23, wenn mit dem Board of Education Rev B gearbeitet wird.

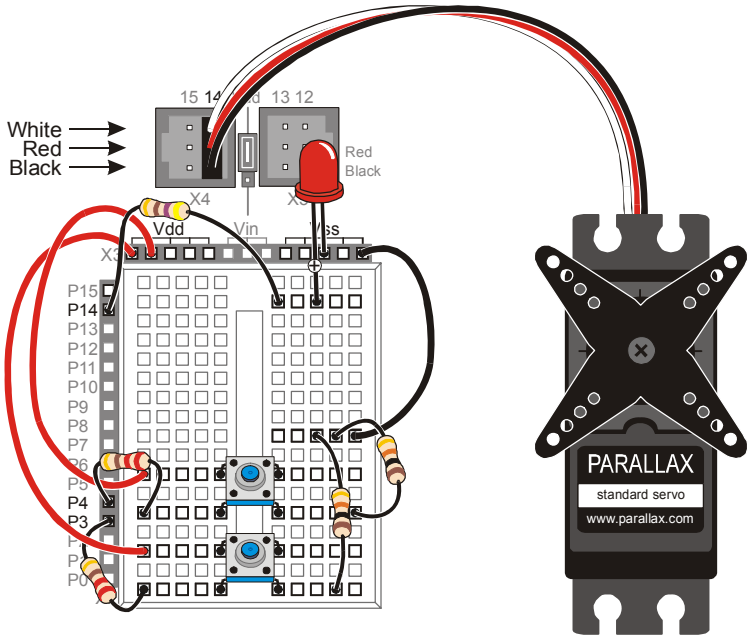


Abbildung 4-21
Board of Education Rev C
Servo Schaltung mit zusätzlichen Taster Schaltungen

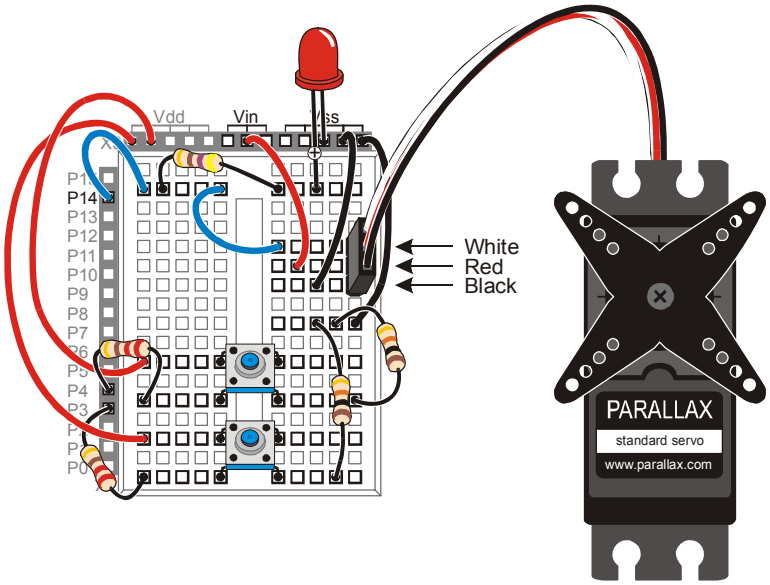


Abbildung 4-22
HomeWork Board Servo Schaltung mit hinzugefügten Tasterschaltungen

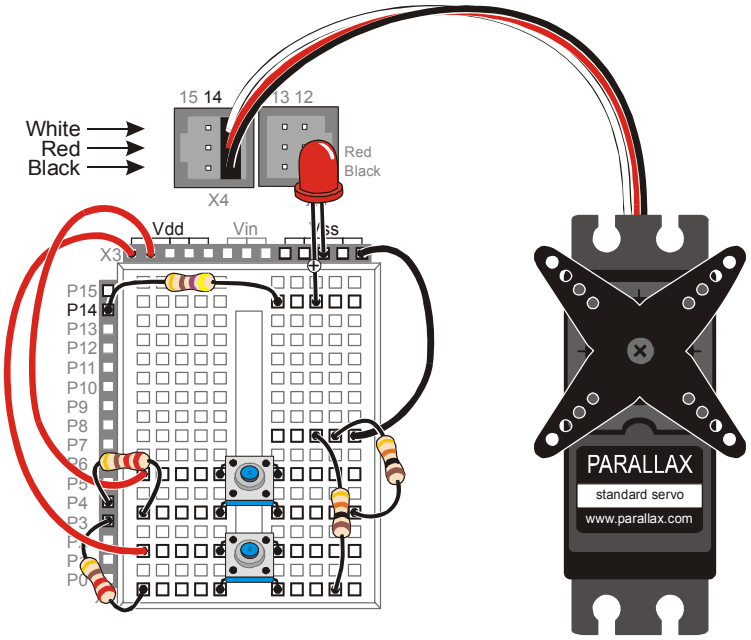


Abbildung 4-23
Board of Education Rev B
Servo Schaltung mit hinzugefügten Taster Schaltungen

√ Teste die mit P3 verbundenen Taster mit dem Originalprogramm ReadPushbuttonState.bs2. Der Abschnitt, in dem das Programm und die Nutzerhinweise zu finden sind, beginnt auf Seite 81.

√ Änder das Programm so, dass es P4 ausliest.

√ Führe das geänderte Programm aus und teste den an P4 angeschlossenen Taster.

Programmierung einer Tastersteuerung für Servos

Die Tastensteuerung eines Servos unterscheidet sich nicht sehr stark von der für eine LED. Mit Hilfe von **`IF…THEN`** Programmblöcken werden die Tasterzustände abgefragt und zum Wert der Variablen **`duration`** entweder addiert oder subtrahiert. Wird einer der beiden Taster gedrückt, erhöht sich der Wert von **`duration`**, wird der andere Taster gedrückt, erniedrigt sich **`duration`**. Eine geschachtelte **`IF…THEN`** Anweisung wird benötigt, um zu überprüfen, ob die Variable **`duration`** zu groß (größer 1000) oder zu klein (kleiner 500) ist.

Programmbeispiel: ServoControlWithPushbuttons.bs2

In diesem Beispielprogramm wird das Steuerhorn entgegen der Uhrzeigerrichtung bewegt, sobald der mit P4 verbundene Taster gedrückt wird. Die Bewegung wird ausgeführt, solange der Taster gedrückt wird und der Wert von **`duration`** kleiner als 1000 ist. Wird der mit P3 verbundene Taster gedrückt, dreht sich der Servo in Uhrzeigerrichtung. Die Drehbewegung eines Servos ist auch begrenzt durch die Variable **`duration,`** deren Wert nicht unter 500 liegen darf. Während des Programmlaufs wird über das Debug Terminal der Wert von **`duration`** angezeigt.

- √ Gib das Programm ServoControlWithPushbuttons.bs2 in den BASIC Stamp Editor ein und führe es aus.
- √ Überprüfe, ob sich der Servo entgegen der Uhrzeigerrichtung dreht, wenn der mit P4 verbundene Taster gedrückt wird.
- √ Überprüfe, ob der Servo mit seiner Bewegung entgegen dem Uhrzeigersinn anhält, sobald **`duration`** > 1000 ist.
- √ Überprüfe, ob der Servo sich in Uhrzeigerrichtung bewegt, wenn der mit P3 verbundene Taster gedrückt wird.
- √ Überprüfe, ob der Servo mit seiner Bewegung in Uhrzeigerrichtung anhält, sobald **`duration`** < 500 ist.

```
' What's a Microcontroller - ServoControlWithPushbuttons.bs2
' Press and hold P4 pushbutton to rotate the servo counterclockwise,
' or press the pushbutton connected to P3 to rotate the servo clockwise.

' {$STAMP BS2}
' {$PBASIC 2.5}

duration       VAR     Word

duration = 750

DO

  IF IN3 = 1 THEN
    IF duration > 500 THEN
      duration = duration - 25
    ENDIF
  ENDIF
```

```
  IF IN4 = 1 THEN
    IF duration < 1000 THEN
      duration = duration + 25
    ENDIF
  ENDIF

  PULSOUT 14, duration
  PAUSE 10

  DEBUG HOME, DEC4 duration, " = duration"

LOOP
```

Du bist dran – Softwarestopper

Servos verfügen über eingebaute mechanische Stopper, die sie daran hindern zu überdrehen. Wird an einen Servo beispielsweise der Befehl **`PULSOUT 14, 2000`** geschickt, dreht er nicht in die im Argument ***`Duration`*** angegebene Position 2000. Ein eingebauter mechanischer Stopper begrenzt die Drehbewegung und schützt damit den Servo. Wenn man vorsichtig das Steuerhorn eines Servos bewegt, fühlt man, wann die mechanischen Stopper einsetzen. Änder das Beispielprogramm in dieser Übung so ab, dass die BASIC Stamp die Drehbewegung des Servos auf einen engeren Bereich begrenzt, als von den mechanischen Stoppern vorgegeben.

- √ Speicher das Programm Save ServoControlWithPushbuttons.bs2 unter einem neuen Namen.
- √ Passe die Softwaregrenzen für die Drehbewegung eines Servos durch die Werte 650 und 850 statt 500 und 1000 an.
- √ Setze den Wert der Variablen **`duration`** auf 10 statt auf 25.
- √ Beschreibe, welche Veränderungen im Verhalten des Servos zu beobachten sind, wenn der Taster gedrückt wird.
- √ Starte das Programm und vergleiche die aktuellen Ergebnisse mit den erwarteten.

ZUSAMMENFASSUNG

In diesem Kapitel hast du rechnergesteuerte Bewegungen eines Servos kennen gelernt. Ein Servo ist eine Art Stellmotor, der sich mit Hilfe elektronischer Signale in eine bestimmte Position bewegen lässt oder eine bestimmte Position beibehält. Die Signale haben die Form von Impulsen mit einer Impulsdauer von 1 - 2 ms. Um den Servo in einer Position zu fixieren, müssen die Impulse alle 20 ms wiederholt werden.

Mit Hilfe des Befehls **PULSOUT** überträgt die BASIC Stamp Signale. Impulse müssen alle 20 ms erneut an einen Servo übertragen werden, damit er seine Position beibehalten kann. In einem Programm wird das dadurch umgesetzt, dass man die Befehle **PULSOUT** und **PAUSE** in einer Schleife anordnet. Der Argumentwert ***Duration*** im Befehl **PULSOUT** kann über eine Variable zugewiesen werden. Das Steuerhorn eines Servos bewegt sich auf diese Weise in einzelnen Schritten.

In diesem Kapitel wurden mehrere Möglichkeiten aufgezeigt, wie man einen Wert in einer Variablen ablegt. Der Wert kann über das Debug Terminal und den Befehl **DEBUGIN** eingelesen werden oder er erfolgt durch eine Wertzuweisung einer Variablen (wir haben dies bei **FOR…NEXT** Schleifen gesehen). Mit dieser Technik führt ein Servo schnelle Drehbewegung aus. Mit Hilfe von **IF…THEN** Anweisungen lassen sich Taster überwachen und durch Addition/Subtraktion zum/vom Argument ***Duration*** des Befehls **PULSOUT** in entsprechende Bewegungen umsetzen. Damit sind wir, je nach Programmaufbau und Einbindung der Taster, in der Lage bei einem Servo Positionsansteuerungen und Beeinflussungen von Drehbewegung vorzunehmen.

Verständnisfragen

1. Wie heißen die vier externen Bauteile eines Servos? Wofür werden sie benötigt?
2. Wird eine LED Schaltung benötigt, um einen Servo anzusteuern?
3. Mit welchem Befehl wird das LOW Signal, das an einen Servo übertragen wird, angesprochen. Mit welchem Befehl wird das HIGH Signal angesprochen?
4. Mit welcher Programmiertechnik lässt sich eine feste Haltezeit eines Servos für eine bestimmte Position umsetzen?
5. Beschreibe mit eigenen Worten, wie über das Debug Terminal an die BASIC Stamp Informationen übertragen werden? Welcher Befehl wird benötigt, damit die BASIC Stamp vom Debug Terminal Mitteilungen empfängt?
6. Mit welcher Programmiertechnik lässt sich die Drehbewegung eines Servos einschränken?

Übungen

1. Schreibe ein Programm, das den **PULSOUT** Wert, der einen Servo mit einer ***Duration*** von 700 auf 800 und wieder zurück auf 700 ansteuert, in einer Schrittweite von (a) 1, (b) 4 durchläuft.
2. Füge zur Antwort in Übung 1b eine geschachtelte **FOR…NEXT** Schleife ein, so dass 10 Impulse übertragen werden, bevor das Argument ***Duration*** des Befehls **PULSOUT** um 4 erhöht wird.

Projekt

1. Änder das Programm ServoControlWithDebug.bs2 so ab, dass es einen General-Aus Taster abfragt. Wird der General-Aus Taster (P3 Taster) gedrückt, sollte das Debug Terminal keine weiteren Anweisungen annehmen und folgende Information ausgeben:“Press Start switch to start machinery”. Wird der Startschalter (P4 Taster) gedrückt, sollte das normale Programm anlaufen. Wird die Spannungsversorgung ab- und wieder angeschaltet, sollte das Programm sich so verhalten, als ob der General-Aus Schalter gedrückt wurde.

Lösungen

Q1. 1. Stecker – Verbindet Servo mit Energie- und Signalquelle.
    2. Kabel – Spannungsversorgung und Signalübertragung
    3. Steuerhorn – Das bewegliche Teil an einem Servo.
    4. Gehäuse – Enthält Gleichstrommotor, Getriebe und Steuerschaltungen.

Q2. Nein, eine LED zeigt nur an, dass Steuersignale empfangen werden.

Q3. LOW Zustände werden mit dem Befehl **PAUSE** gesteuert. HIGH Zustände werden mit dem Befehl **PULSOUT** gesteuert.

Q4. **FOR...NEXT** Schleife.

Q5. Klicke in das weiße Fenster über dem Mitteilungsfenster des Debug Terminals und gib eine Mitteilung ein, die übertragen werden soll. Benutze den Befehl **DEBUGIN**, um die eingegebenen Zeichen an die BASIC Stamp zu übertragen.

Q6. Mit einer geschachtelten **IF...THEN** Anweisung.

E1. a)

```
FOR counter = 700 TO 800
  PULSOUT 14, counter
  PAUSE 20
NEXT
FOR counter = 800 TO 700
  PULSOUT 14, counter
  PAUSE 20
NEXT
```

b) Add "**STEP 4**" to both **FOR** loops.

```
FOR counter = 700 TO 800 STEP 4
  PULSOUT 14, counter
  PAUSE 20
NEXT
FOR counter = 800 TO 700 STEP 4
  PULSOUT 14, counter
  PAUSE 20
NEXT
```


E2.

```
FOR counter = 700 TO 800 STEP 4
  FOR pulses = 1 TO 10
    PULSOUT 14, counter
    PAUSE 20
  NEXT
NEXT
FOR counter = 800 TO 700 STEP 4
  FOR pulses = 1 TO 10
    PULSOUT 14, counter
    PAUSE 20
  NEXT
NEXT
```

P1. Es gibt verschiedene Lösungswege; eine mögliche Lösung ist:

```
' What's a Microcontroller - Ch04Prj01Soln2__KillSwitch.bs2
' Send messages to the BASIC Stamp to control a servo using
' the Debug Terminal as long as kill switch is not being pressed.

' Contributed by: Professor Clark J. Radcliffe, Department
' of Mechanical Engineering, Michigan State University

' {$STAMP BS2}
' {$PBASIC 2.5}

counter VAR Word
pulses VAR Word
duration VAR Word
DO
  PAUSE 2000
  IF (IN3 = 1) AND (IN4 = 0) THEN
    DEBUG "Press Start switch to start machinery.     ", CR ,CRSRUP
  ELSEIF (IN3 = 0) AND (IN4 = 1) THEN
    DEBUG CLS, "Enter number of pulses:", CR
    DEBUGIN DEC pulses

    DEBUG "Enter PULSOUT duration:", CR
    DEBUGIN DEC duration
```

```
    DEBUG "Servo is running...", CR

    FOR counter = 1 TO pulses
      PULSOUT 14, duration
      PAUSE 20
    NEXT

    DEBUG "DONE"
    PAUSE 2000
  ENDIF
LOOP
```

Weiterführende Bücher

Der Servo und die Anwendung von sensorgesteuerten Servos wird ausführlicher in einer Reihe von Stamps in Class Büchern behandelt.

"*Advanced Robotics: with the Toddler*", Student Guide, Version 1.2, Parallax Inc., 2003

Advanced Robotics: with the Toddler steuert mit Servos die Beinbewegungen des Toddlers von Parallax. Obwohl das Gehen eine alltägliche Beschäftigung ist, ist es eine technische Herausforderung, einer Maschine das Laufen, Erkennen und Ausweichen von Hindernissen und das Halten von Gleichgewichten beizubringen. Mit diesem laufenden Roboter wenden wir uns an fortgeschrittene Entdecker, die die Lehrgänge *What's a Microcontroller?* und *Robotics mit dem Boe-Bot* oder *SumoBot* erfolgreich absolviert haben.

"*Robotics with the Boe-Bot*", Student Workbook, Version 2.0 Parallax Inc., 2003

Der englische Text liegt in einer deutschen Übersetzung unter dem Titel *„Robotics mit dem Boe-Bot"* als pdf-Datei im Internet vor.

Robotics mit dem Boe-Bot stützt sich auf die gleichen Steuerungsprinzipien beim Servo wie die, die wir gerade besprochen haben; Servos lassen sich aber auch als Motoren in Robotern betreiben. Mit Hilfe der BASIC Stamp, dem Board of Education und dem Bausatz *Robotics with the Boe-Bot* führt dieses Buch in die Grundsätze der Robotersteuerung ein, erläutert die Navigation mit Hilfe von Sensoren und stellt eine Reihe von tiefergehenden Themen vor, die sich mit dem Lösen von Problemen mit Hilfe von künstlicher Intelligenz und der Navigation über einfache Steuerungssysteme befassen.

"*SumoBot*", Student Workbook, Version 1.1, Parallax Inc., 2002

Robot Sumo ist ein sehr aufregender Wettbewerb mit hohem Spaßfaktor für Beobachter und Teilnehmer. *SumoBot* dokumentiert Bau, Test und Kampf mit einem eigenständigen Mini-Sumo aus der SumoBot® Roboterklasse. Dieses Buch ist eine verdichtete Darstellung des Buches *Robotics mit dem Boe-Bot* mit dem Ziel, auf einen von Robotern geführten Sumowettkampf vorzubereiten und diesen Wettkampf auch zu gewinnen.

Kapitel 5: Drehbewegungen auswerten

EINSTELLKNÖPFE FÜR DIE GERÄTESTEUERUNG

In vielen Haushalten wird die Lichtstärke im Raum über einen Dimmer eingestellt. Wird der Drehknopf beispielsweise nach rechts gedreht, wird die Beleuchtung stärker, bei einer Linksdrehung nimmt sie ab. Modelleisenbahnen werden über einen Trafo-Drehknopf in ihrer Geschwindigkeit und dem Vor- bzw. Rückwärtslauf gesteuert. Viele Geräte verfügen über Einstellknöpfe und Kurbeln, mit denen sich zum Beispiel die Schneidemesser einer Maschine positionieren lassen

Radios und Verstärker verfügen über Einstellknöpfe; mit ihnen lassen sich z.B. Lautstärke und Klang regulieren. Abbildung 5-1 zeigt ein einfaches Beispiel eines Lautstärkereglers. Durch Drehen des Knopfes wird die Lautstärke des Gerätes verändert. Schaltungen mit ähnlicher Wirkung finden sich in Joysticks und auch im Servo aus Kapitel 4: Servosteuerung.

Abbildung 5-1
Regelung der Lautstärke an einem Lautsprecher

DAS POTENTIOMETER - EIN VERÄNDERLICHER WIDERSTAND

Das Bauteil, das sich hinter dem Einstellknopf, in einem Joystick und Servo versteckt, heißt Potentiometer oder sprachlich verkürzt: Poti. Abbildung 5-2 zeigt ein Bild verschiedener Potentiometer. Beachte, dass sie alle über drei Anschlusspins verfügen.

Abbildung 5-2
Einige Beispiele
für Potentiometer

In Abbildung 5-3 wird das Schaltungssymbol und die Bauteildarstellung des in diesem Kapitel benutzten Potentiometers gezeigt. Zwischen den Anschlüssen A und B liegt ein Widerstandselement von 10 kΩ Widerstand. Anschluss W ist mit einem Schleiferkontakt verbunden, der mit dem Widerstandselement irgendwo zwischen den Punkten A und B Kontakt hat.

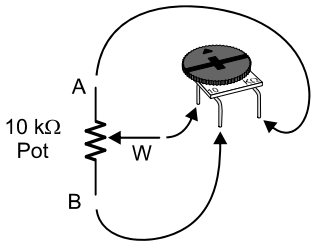


Abbildung 5-3
Potentiometer
Schaltungssymbol und
Bauteil Darstellung

Abbildung 5-4 zeigt, wie der Schleifkontakt eines Potentiometers arbeitet. Wird der Einstellknopf gedreht, fährt der Schleifkontakt über die Widerstandsstrecke. Bei Drehung in Zeigerrichtung nähert sich der Schleifer dem Punkt A, bei Drehung in entgegengesetzter Richtung dem Punkt B.

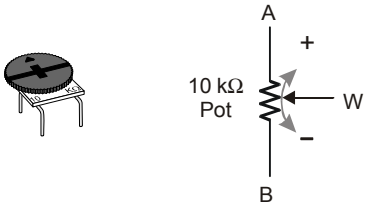


Abbildung 5-4
Bewegen des
Schleifkontaktes eines
Potentiometers

ÜBUNG 1: AUFBAU UND TEST EINER POTENTIOMETERSCHALTUNG

Schaltet man unterschiedlich große Widerstände in Reihe mit einer LED, werden unterschiedlich starke Ströme im Stromkreis fließen. Ein großer Widerstand in einer LED Schaltung führt zu einem geringen Stromfluss mit der Folge, dass die LED nur schwach leuchtet. Ein kleiner Widerstand erhöht die Stromstärke und lässt die LED stärker leuchten. Verbindet man die Anschlüsse W und A eines Potentiometer, das in Reihe mit einer LED geschaltet ist, lässt sich der Widerstand und damit auch die Helligkeit einer LED verändern. In dieser Übung wird das Potentiometer als variabler Widerstand zur Helligkeitssteuerung einer LED eingesetzt.

Bauteile: Potentiometerschaltung

1 Potentiometer – 10 kΩ
1 Widerstand – 220 Ω (rot-rot-braun)
1 LED – Farbe beliebig
1 Drahtbrücke

Aufbau der Testschaltung Potentiometer

Abbildung 5-5 zeigt eine Schaltung, mit der sich die Helligkeit einer LED über ein Potentiometer verändern lässt.

√ Baue die Schaltung nach Abbildung 5-5 auf.

Tipp: Begradige mit einer Flachzange die Anschlussbeinchen des Potentiometers, bevor es in das Steckbrett gesteckt wird. Gerade Anschlussbeine garantieren einen besseren Kontakt mit den Steckbrettbuchsen.

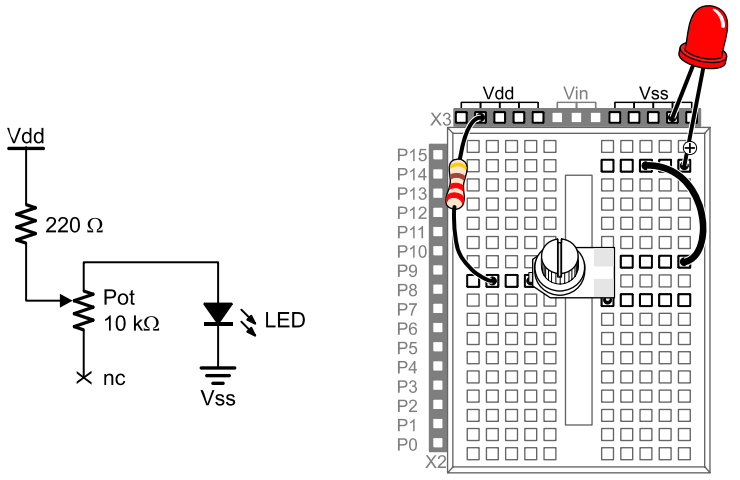


Abbildung 5-5
Testschaltung
Potentiometer

Test der Potentiometerschaltung

√ Drehe das Potentiometer bis zum Anschlag in Uhrzeigerrichtung (siehe Abbildung 5-6).

Vorsicht: Lässt sich dein Potentiometer nicht so weit drehen, wie in der Abbildung gezeigt, dann erzwinge es nicht, sonst zerstörst du das Bauteil. Drehe den Einstellknopf gerade so weit, bis ein Weiterdrehen nicht mehr möglich ist.

√ Bewege den Drehknopf langsam in entgegengesetzter Uhrzeigerrichtung in die in Abbildung 5-6 (b), (c), (d), (e), und (f) gezeigten Positionen und beobachte, wie hell die LED bei jeder Einstellung aufleuchtet.

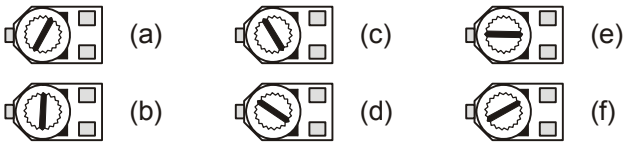


Abbildung 5-6
Potentiometer
Einstellschraube

(a) … (f) zeigen unterschiedliche Schleiferstellungen.

Arbeitsweise der Potentiometerschaltung

Der Gesamtwiderstand in der Testschaltung beträgt 220 Ω plus dem Widerstandswert zwischen den Punkten A und W des Potentiometers. Sein Wert liegt irgendwo zwischen 0 und 10 kΩ. Wird an der Einstellschraube eines Potentiometers gedreht, verändert sich der Widerstand der Strecke zwischen A und W. Damit ändert sich gleichzeitig der Stromfluss durch die LED.

ÜBUNG 2: WIDERSTANDSBESTIMMUNG DURCH ZEITMESSUNG

In dieser Übung lernen wir ein neues Bauteil kennen: den Kondensator. Ein Kondensator verhält sich wie eine wiederaufladbare Batterie, die ihre Ladung allerdings nur für einen kurzen Zeitraum speichern kann. In dieser Übung erfährst du auch etwas über die sogenannte Entladezeit eines Kondensators; es ist die Zeit, die vergeht, bis sich ein Kondensator C über einen Widerstand R entladen hat. Um uns mit dieser neuen Größe vertraut zu machen, werden wir mit verschieden großen Widerständen und Kondensatoren die Entladezeit messen. In dieser Übung wird die BASIC Stamp so programmiert, dass sie zunächst einen Kondensator auflädt und anschließend die Zeit misst, in der sich der Kondensator über einen Widerstand entlädt.

Einführung Kondensator

Abbildung 5-7 zeigt die Schaltung und Bauteilzeichnung eines Elektrolytkondensators, so wie wir ihn in dieser Übung benutzen. Die Kapazität eines Kondensator wird in der Einheit Microfarad (µF) angegeben und ist i.a. auf das Bauteil aufgedruckt.

Dieser Kondensator (Elektrolytkondensator) hat einen positiven (+) und negativen (-) Pol. Der negative Pol ist auf dem Zylinder mit einem Minuszeichen markiert, ein kurzer Anschlussdraht führt aus dem Zylinder heraus. Achte bitte immer darauf, dass die Anschlussdrähte genau so, wie in der Schaltung angegeben, im Steckbrett positioniert werden. Bei Vertauschung der Anschlusspunkte wird das Bauteil zerstört. Bei Schaltungen mit höheren Betriebsspannungen kann der falsche Einbau eines Elektrolytkondensators zur Explosion des Bauteils führen.

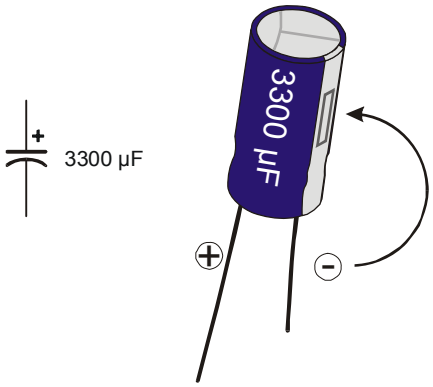


Abbildung 5-7
3300 µF Kondensator Schaltungsaufbau und Bauteilzeichnung

Achte auf die unterschiedlich langen Anschlussdrähte des Kondensators und die Zuweisung der Polung.

Bauteile der Schaltung

1 Elektrolytkondensator – 3300 µF
1 Elektrolytkondensator – 1000 µF
1 Widerstand – 220 Ω (rot-rot-braun)
1 Widerstand – 470 Ω (gelb-violett-braun)
1 Widerstand – 1 kΩ (braun-schwarz-rot)
1 Widerstand – 2 kΩ (rot-schwarz-rot)
1 Widerstand – 10 kΩ (braun-schwarz-orange)

Benötigte Ausstattung: Sicherheitsbrille.

Aufbau und Test einer RC-Schaltung

Abbildung 5-8 zeigt den Schaltplan und Abbildung 5-9 den Schaltungsaufbau für diese Übung. Es werden Zeitmessungen mit verschiedenen Widerstandswerten R_i durchgeführt.

Sicherheitshinweise

Bevor ein Kondensator von 3300 µF Kapazität in eine Schaltung eingebaut wird, achte unbedingt auf die angegebene Polung des Bauteils. Beachte, dass der mit dem negativen Pol verbundene Anschlussdraht mit dem Metallkörper des Kondensators verbunden ist und sich in Richtung des aufgedruckten Minuszeichens (-) befindet. Suche in Abbildung 5-7 die beiden Anschlüsse (+) und (-).

Ein 3300 µF Kondensator wird problemlos arbeiten, wenn sichergestellt ist, dass er mit korrekter Polung angeschlossen ist, so wie in Abbildung 5-8 und Abbildung 5-9 gezeigt.

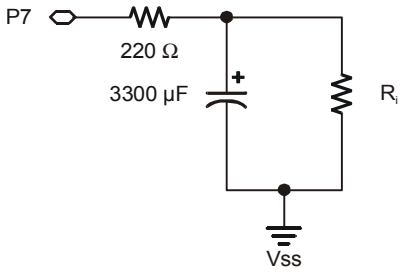
VERTAUSCHE bei einem Elektrolytkondensator NIEMALS die Polarität. Das am (+)-Pol anliegende Potential muss immer höher sein, als das am (-)-Pol anliegende. Vss kennzeichnet das niedrigere Potential auf dem Board of Education und dem BASIC HomeWork Board. Der Kondensator ist korrekt gepolt, wenn der negative Pol des Kondensators mit Vss verbunden ist.

Setze während der Übung eine Schutzbrille auf.

Schalte die Spannungsquelle ab, wenn du eine Schaltung aufbaust oder veränderst.

Komme mit dem Gesicht oder den Händen nicht in die Nähe eines Kondensators, wenn eine Spannungsquelle angeschlossen ist.

√ Schalte die Spannungsversorgung ab und baue die abgebildete Schaltung zunächst mit einem R_i = 470 Ω Widerstand auf.



R_1 = 470 Ω
R_2 = 1 kΩ
R_3 = 2 kΩ
R_4 = 10 kΩ

Abbildung 5-8
Schaltbild einer RC-Schaltung

Für R_i sind in der Schaltung vier verschiedene Widerstandswerte vorgesehen. Starte mit R_i = 470 Ω und wähle dann R_i = 1 kΩ, etc.

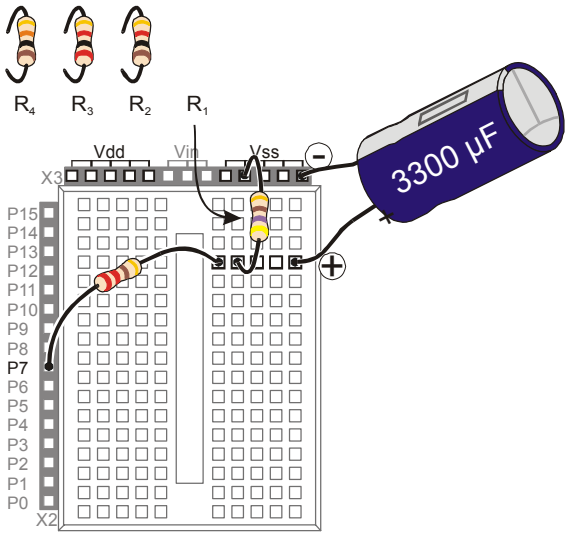


Abbildung 5-9
Aufbau der Schaltung aus Abbildung 5-8

Achte darauf, dass der negative Pol des Kondensators so in das Board gesteckt wird, wie in dieser Abbildung. Der negative Pol ist mit Vss verbunden.

Entladezeit einer RC-Schaltung

Obwohl die Entladezeit eines Kondensators mit einer Stoppuhr gemessen werden kann, lässt sich dieser Vorgang bei höherer Messgenauigkeit automatisieren und über die programmierte BASIC Stamp durchführen.

Programmbeispiel: PolledRcTimer.bs2

- √ Gibt das Programm PolledRcTimer.bs2 ein und führe es auch.
- √ Beobachte, wie die BASIC Stamp den Kondensator lädt und anschließend die Entladezeit misst.
- √ Trage die Messzeit in der 470 Ω Zeile der Tabelle 5-1 ein (die Entladezeit des Kondensators).
- √ Trenne die Spannungsversorgung vom Board of Education oder BASIC Stamp HomeWork Board.
- √ Ersetze den 470 Ω Widerstand R_i in Abbildung 5-8 und Abbildung 5-9 auf Seite 153 durch einen 1 kΩ Widerstand.
- √ Schließe die Spannungsquelle wieder an.
- √ Trage die gemessene Zeit ein (für den 1 kΩ Widerstand).
- √ Wiederhole die Schritte für jeden Widerstand aus Tabelle 5-1.

Tabelle 5-1: Widerstand und Entladezeit mit C = 3300 µF	
Widerstand (Ω)	**Gemessene Zeit (s)**
470	
1 k	
2 k	
10 k	


```
' What's a Microcontroller - PolledRcTimer.bs2
' Reaction timer program modified to track an RC-time voltage decay.

' {$STAMP BS2}
' {$PBASIC 2.5}

timeCounter    VAR     Word
counter        VAR     Nib

DEBUG CLS

HIGH 7
DEBUG "Capacitor Charging...", CR

FOR counter = 5 TO 0
  PAUSE 1000
  DEBUG DEC2 counter, CR, CRSRUP
NEXT

DEBUG CR, CR, "Measure decay time now!", CR, CR
INPUT 7

DO

  PAUSE 100
  timeCounter = timeCounter + 1

  DEBUG ? IN7
  DEBUG DEC5 timeCounter, CR, CRSRUP, CRSRUP

LOOP UNTIL IN7 = 0

DEBUG CR, CR, CR, "The RC decay time was ",
      DEC timeCounter, CR,
      "tenths of a second.", CR, CR

END
```

Arbeitsweise von PolledRcTimer.bs2

Er werden in diesem Programm zwei Variable deklariert. In der Variablen **timeCounter** wird die Zeit abgespeichert, die der Kondensator benötigt, um sich über R_i zu entladen. **counter** wird als Zähler eingesetzt, solange sich der Kondensator auflädt.

```
timeCounter     VAR     Word
counter         VAR     Nib
```

Der Befehl **DEBUG CLS** löscht das *Debug Terminal*. **HIGH 7** setzt P7 auf HIGH und startet den Aufladevorgang des Kondensators; die Mitteilung **"Capacitor charging…"** wird angezeigt. In der **FOR…NEXT** Schleife wird ein Rückwärtszähler eingesetzt; während er auf Null zählt, wird der Kondensator geladen und die Spannung an seinen Polen wächst , je nach eingesetztem Ladewiderstand R_i, auf Werte zwischen 2,5 bis 4,9 V an.

```
DEBUG CLS

HIGH 7
DEBUG "Capacitor Charging...", CR

FOR counter = 5 TO 0
  PAUSE 1000
  DEBUG DEC2 counter, CR, CRSRUP
NEXT
```

In einer Nachricht wird angezeigt, dass die Entladezeit jetzt gemessen wird.

```
DEBUG CR, CR, "Measure decay time now!", CR, CR
```

Während der Kondensator sich über den Widerstand R_i entlädt, wird der I/O Pin von **HIGH** auf **INPUT** umgestellt. Sobald der I/O Pin die Schaltung freigegeben hat, entlädt sich der Kondensator über den Widerstand und die an C anliegende Spannung nimmt rasch ab.

```
INPUT 7
```

Im Kapitel über die Taster haben wir mit der BASIC Stamp über die Variablen **IN3** und **IN4** HIGH und LOW Signale erkannt. Als HIGH Signal wurde bisher das Potential Vdd definiert, als LOW das Potential Vss. Zukünftig werden wir Signale mit einem Potential über 1,4 V als HIGH ansprechen. Es kann damit natürlich auch 5 V betragen. Ein LOW Signal weist dementsprechend ein Potential zwischen 1,4 V und 0 V auf. **DO…LOOP** fragt P7 alle 100 ms ab, bis **IN7** von 1 auf 0 springt. In diesem Moment hat die Entlade-spannung des Kondensators einen Wert unterhalb 1,4 V erreicht.

```
DO
```

```
    PAUSE 100
    timeCounter = timeCounter + 1

    DEBUG ? IN7
    DEBUG DEC5 timeCounter, CR, CRSRUP, CRSRUP

  LOOP UNTIL IN7 = 0
```

Das Ergebnis wird angezeigt und das Programm wird beendet.

```
  DEBUG CR, CR, CR, "The RC decay time was ",

        DEC timeCounter, CR,
        "tenths of a second.", CR, CR

  END
```


Du bist dran – Eine schnellere Schaltung

Wird ein neuer Kondensator mit nur einem Drittel der ursprünglichen Kapazität in die Schaltung eingebaut, verkürzt sich die Entladezeit für jeden eingesetzten Widerstand auf ein Drittel. In Übung 3 wird ein Kondensator mit 10.000-mal kleinerer Kapazität benutzt; die BASIC Stamp übernimmt wieder die Zeitmessung. Wir bedienen uns dazu des Befehls **`RCTIME`**.

- √ Trenne das Board of Education oder HomeWork Board von der Energieversorgung.
- √ Ersetze den 3300 µF Kondensator durch einen 1000 µF Kondensator.
- √ **Überzeuge dich davon, dass die Polung des Kondensator korrekt ist.** Der negative Anschluss wird mit Vss verbunden.
- √ Lege die Spannungsversorgung wieder an.
- √ Wiederhole die Schritte im Beispielprogramm PolledRcTimer.bs2.bs2 und notiere die Zeitmessungen in Table 5-2.
- √ Vergleiche die Zeitmessungen mit denen aus Tabelle 5-1 (3300 µF). Lässt sich die Aussage bestätigen, dass bei 1/3 Kapazitätswert auch die Entladezeit auf 1/3 sinkt?

Table 5-2: Widerstand und Entladezeit für C = 1000 µF	
Widerstand (Ω)	**Gemessene Zeit (s)**
470	
1 k	
2 k	
10 k	

ÜBUNG 3: POTENTIOMETERWERTE MIT DER BASIC STAMP AUSLESEN

In Übung 1 wurde ein Potentiometer als veränderbarer Widerstand eingesetzt. Sein Widerstandswert hing von der Position des Drehreglers ab. In Übung 2, wurde eine RC-Schaltung benutzt, um verschiedene Widerstände zu bestimmen. In dieser Übung wird eine RC-Schaltung aufgebaut, mit der ein Potentiometerwert bestimmt werden kann. Für die Zeitmessung wird die BASIC Stamp eingesetzt. Die Kapazität ist sehr klein, die zugehörige Zeitmessung deshalb sehr kurz (einige Millisekunden). Die BASIC Stamp wird trotzdem hervorragende Werte zur Bestimmung des Widerstandes zwischen den Anschlusspunkten A und W des Potentiometers liefern.

Bauteile: RC-Schaltung und BASIC Stamp

1 Potentiometer – 10 kΩ
1 Widerstand – 220 Ω (red-red-brown)
2 Verbindungsdrähte
1 Kondensator – 0.1 µF , wie in Abbildung 5-10 gezeigt.
1 Kapazität – 0.01 µF, wie in Abbildung 5-10 zu sehen.
2 Verbindungsdrähte

Diese Kondensatoren haben keine ausgewiesenen + und – Pole. Bei diesen Kondensatoren muss nicht auf die korrekte Polarität beim Einbau geachtet werden.

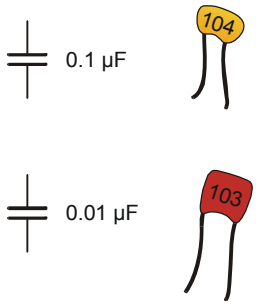


Abbildung 5-10
Keramischer Kondensator

Der 0.1 µF Kondensator (oben) und der 0.01 µF Kondensator (unten) sind beide nicht-polar. Sie haben keine ausgewiesenen Pole.

Aufbau einer RC-Schaltung für die BASIC Stamp

Abbildung 5-11 zeigt eine RC-Schaltung und Abbildung 5-12 den Schaltungsaufbau. Mit dieser Schaltung wird mit der BASIC Stamp und einem PBASIC Programm der Widerstandswert des Potentiometers ausgelesen.

√ Baue die Schaltung nach Abbildung 5-11 auf.

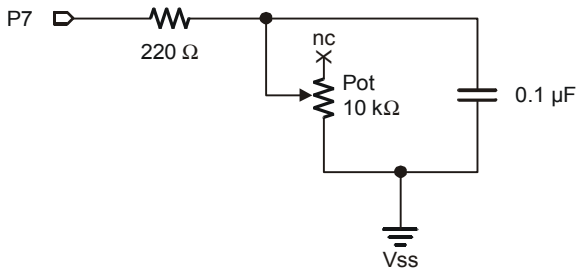


Abbildung 5-11
RC-Schaltung mit Potentiometer

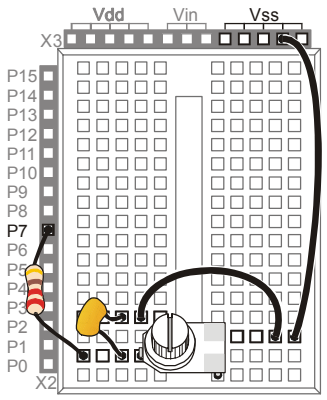


Abbildung 5-12
Verdrahtungsplan nach der Schaltung aus Abbildung 5-11

Programmierung: Messung der Entladezeit

Das BASIC Programm aus Übung 2 überprüfte alle 100 ms, ob **`IN7 = 0`** ist und bestimmte über die Anzahl der Abfragen die Entladezeit. Fällt **`IN7`** von 1 auf 0, ist die am Kondensator anliegende Spannung unter 1,4 Volt abgesunken und der Wert von **`timeCounter`** gab in der Einheit Zehntelsekunden die Entladezeit an, in der die Kondensatorspannung unter 1,4 Volt abgesunken war. Im folgenden Beispielprogramm wird mit dem PBASIC Befehl **`RCTIME`** die Entladezeit in Einheiten von 2 µs aufgenommen. Statt mit einer Genauigkeit von Zehntelsekunden wird über **`RCTIME 7, 1, time`** die Zeit in der Variablen **`time`** in Einheiten von jeweils Zweimillionstel Sekunden abgelegt. Auf Grund dieser hohen Auflösung kann die Kapazität des Kondensators von 3300 µF auf 0,1 oder sogar 0,01 µF verkleinert werden. Bei jeder Veränderung der Potentiometereinstellung ist die über **`RCTIME`** bestimmte Zeit ein Maß für die Position des Schleifkontaktes und damit des Widerstandswertes.

Programmbeispiel: ReadPotWithRcTime.bs2

- √ Gib das Programm ReadPotWithRcTime.bs2 ein und starte es.
- √ Verändere durch Drehen des Einstellreglers den Widerstand des Potentiometers und beobachte dabei den Wert der Variablen **`time`** auf dem Debug Terminal.

```
' What's a Microcontroller - ReadPotWithRcTime.bs2
' Read potentiometer in RC-time circuit using RCTIME command.

' {$STAMP BS2}
' {$PBASIC 2.5}
```

```
time VAR Word

DO

  HIGH 7
  PAUSE 100
  RCTIME 7, 1, time
  DEBUG HOME, "time = ", DEC5 time

LOOP
```


Arbeitsweise von ReadPotWithRcTime.bs2

Der Pseudocode stellt die abzuarbeitenden Schritte bei der Bestimmung der Entladezeit dar.

- *Die Variable* **`time`** *als Speicher für die Zeitmessung deklarieren.*
- *Programmblock innerhalb* **`DO…LOOP`** *Schleife:*
  - *Setze I/O Pin P7 auf* **`HIGH`**.
  - *Warte 100 ms (20 ms bis zum Aufladen des Kondensators, 80 ms zum aktivieren des Debug Terminals).*
  - *Führe den Befehl* **`RCTIME`** *aus.*
  - *Speicher die Zeitmessung in der Variablen* **`time`**.
  - *Zeige den Wert von* **`time`** *auf dem Debug Terminal an.*

Bevor der Befehl **`RCTIME`** ausgeführt wird, muss der Kondensator voll aufgeladen sein. Wird **`RCTIME`** anschließend ausgeführt, wechselt die BASIC Stamp die Belegung des I/O Pins von Ausgang auf Eingang. In der Belegung als Eingang ist der I/O Pin in der Schaltung mit einem geöffneten Taster vergleichbar (offener Schaltkreis), siehe Übung 2. Der Befehl **`RCTIME`** wird bei einer Highspeed-Zeitmessung eingesetzt; er misst die Zeit, die vergeht, bis der Kondensator so viel an Ladung verloren hat, dass das Potential am I/O Pin unter die Schwelle von 1,4 V fällt. Statt wie bisher alle 100 ms das Potential zu überprüfen, werden mit dem **`RCTIME`** Befehl die I/O Pins alle 2 µs überprüft.

Du bist dran – Entladezeit verändern

√ Ersetze den 0.1 µF Kondensator durch einen von 0.01 µF.

√ Versuche mit den gleichen Potentiometereinstellungen wie in der ersten Übung zu arbeiten und vergleiche die angezeigten Werte auf dem Debug Terminal mit denen der Kapazität von 0,1 µF. Sind die **`RCTIME`** Messwerte auf ein Zehntel gesunken?

- √ Benutze wieder die Kapazität von 0.1 µF.
- √ Notiere für beide Kapazitäten die höchsten und niedrigsten Messwerte als Vorgriff auf die nächste Übung.

ÜBUNG 4: SERVOSTEUERUNG MIT POTENTIOMETER

Mit Potentiometern und Servos lassen sich einige interessante Dinge anstellen. Dies wäre dann der Einstieg in den Modellbau von Flugzeugen, Autos und Schiffen. In dieser Übung erfährst du, wie man mit der BASIC Stamp eine Potentiometerschaltung überwacht und die Position eines Servos steuert.

Als Beispiel siehst du in Abbildung 5-13 ein ferngesteuertes Modellflugzeug. Es verfügt über Servos für alle Klappen und die Drosseleinstellung des Motors. Die Servos werden über eine Funk-Fernsteueranlage (RC) , die vor dem Flugzeug liegt, gesteuert. Unter jedem Joystick der RC-Anlage befinden sich Potentiometer, mit denen die Servos angesteuert werden, die wiederum das Höhen- und die Seitenruder steuern.

Abbildung 5-13
Modellflugzeug mit Funk-Fernsteuerung

Wie man mit einer RC Anlage ein Flugzeug steuert: Die Potentiometer unter den Joysticks werden von einer Schaltung überwacht, die ihre jeweilige Position in Steuerimpulse für den Servo umsetzt. Diese werden dann in Radiosignale umgewandelt und über den tragbaren Sender an einen Empfänger im Modellflugzeug übertragen. Der Empfänger wandelt diese Signale zurück in Steuerimpulse, die dann den Servo positionieren.

Bauteile: Servosteuerung mit Potentiometer

1 Potentiometer – 10 kΩ
1 Widerstand – 220 Ω (rot-rot-braun)
1 Kondensator – 0.1 µF
1 Parallax Standard Servo
1 LED – beliebige Farbe
2 Verbindungsdrähten

Benutzer eines HomeWork Board benötigen zusätzlich:

1 3-Pin m-m Stecker
4 Verbindungsdrähte

ACHTUNG: Benutze für die Übungen dieses Buches nur den Parallax Standard Servo!

Der Parallax Freilaufservo ist für diese Schaltungen nicht geeignet und könnte bei einem Einsatz beschädigt werden. Ebenso empfehlen wir, keine Servos von Fremdherstellern einzusetzen, da die Spannungsanpassungen in den Versuchsschaltungen nicht gewährleistet sind.

Aufbau von Potentiometer- und Servoschaltung

Diese Übung verwendet zwei bereits bekannte Schaltungen: die Potentiometerschaltung aus der vorherigen Übung und die Servo-Schaltung aus einem früheren Kapitel.

√ Lasse die RC-Schaltung mit dem Potentiometer aus Übung 3 aufgebaut. Musst du sie neu aufbauen, halte dich an Abbildung 5-11 auf Seite 159 und Abbildung 5-12 auf Seite 160. Benutze den 0.1 µF Kondensator.

√ Füge die Servo-Schaltung aus Kapitel 4, Übung 1 hinzu. Denke bitte daran, dass die Schaltung unterschiedlich aufgebaut werden muss, je nach eingesetztem Board:

- Seite 111 – Board of Education Rev C
- Seite 117 – Board of Education Rev B
- Seite 114 – BASIC Stamp HomeWork Board

Programmierung einer Servosteuerung mit Potentiometer

Für das Programm werden der größte und kleinste Messwert der Variablen **`time`** benötigt, die mit dem 0,1 µF Kondensator in der letzten Übung bestimmt wurden.

√ Solltest du den Abschnitt "Du bist dran …" aus der vorherigen Übung noch nicht abgeschlossen haben, gehe noch einmal zurück und arbeite die fehlenden Teile durch.

Für das folgende Beispiel findest du hier die Werte von **`time`** , die ein Techniker von Parallax gemessen hat; deine Messwerte können davon etwas abweichen:

- Rechter Anschlag: 1
- Linker Anschlag: 691

Wie lassen sich nun diese Eingabewerte auf die Werte von 500 und 1000 anpassen, die für die Steuerung eines Servos mit dem **`PULSOUT`** Befehl notwendig sind? Die Antwort ist einfach: durch Multiplikation und Addition! Multipliziere zuerst die Eingabewerte mit einem festgelegten Faktor, so dass die Differenz zwischen der Uhrzeigerrichtung (Minimum) und Gegenuhrzeigerrichtung (Maximum) etwa 500 beträgt. Füge dann eine Konstante hinzu, so dass sich der Wertebereich von 1 … 500 nach 500 … 1000 verschiebt. In der Elektronik spricht man von Skalierung und Offset .

Die Skalierung wird hier noch einmal vorgerechnet:

$$time(maximum) = 691 \times \frac{500}{691} = 691 \times 0.724 = 500$$

$$time(minimum) = 1 \times \frac{500}{691} = 0.724$$

Nach der Skalierung erfolgt der Offset-Schritt (Addition).

$$time(maximum) = 500 + 500 = 1000$$

$$time(minimum) = 0.724 + 500 = 500$$

Der auf Seite 99 eingeführte Operator **`*/`** ist in PBASIC eingebaut und wird für das Skalieren mit rationalen Werten, wie z.B. 0,742, benötigt. Hier siehst du noch einmal, wie **`*/`** aufgerufen wird:

1. Platziere den Wert der Variablen, die mit einer rationalen Zahl multiplizieren werden soll, vor den */ Operator.

   ```
   time = time */
   ```

2. Multipliziere die rationale Zahl mit 256.

   $$new\,fractional\,value = 0.724 \times 256 = 185.344$$

3. Runde die sich ergebende Zahl auf Einer.

   $$new\,fractional\,value = 185$$

4. Platziere diesen Wert nach dem */ Operator.

   ```
   time = time */ 185
   ```

Damit haben wir die Skalierung abgeschlossen; jetzt muss noch ein Offset von 500 hinzugefügt werden. Mit einem zweiten Befehl wird zu **`time`** 500 addiert:

```
time = time */ 185
time = time + 500
```

Jetzt kann **`time`** in das Argument ***`Duration`*** des Befehl **`PULSOUT`** eingesetzt werden.

```
time = time */ 185        ' Scale by 0.724.
time = time + 500         ' Offset by 500.
PULSOUT 14, time          ' Send pulse to servo.
```

Programmbeispiel: ControlServoWithPot.bs2

√ Gib das Programm ein und führe es aus. Verstelle das Potentiometer und überzeuge dich, dass die Bewegung des Servos der Bewegung des Potentiometers folgt.

```
' What's a Microcontroller - ControlServoWithPot.bs2
' Read potentiometer in RC-time circuit using RCTIME command.
' Scale time by 0.724 and offset by 500 for the servo.

' {$STAMP BS2}
' {$PBASIC 2.5}

DEBUG "Program Running!"
```

```
time           VAR     Word

DO
  HIGH 7
  PAUSE 10
  RCTIME 7, 1, time
  time = time */ 185                          ' Scale by 0.724 (X 256 for */).
  time = time + 500                           ' Offset by 500.
  PULSOUT 14, time                            ' Send pulse to servo.
LOOP
```

Du bist dran – Skalierung von Servo und Poti

Potentiometer und Kondensator können bei der eigenen **time** Bestimmung etwas von den Werten abweichen, die in dieser Übung genannt wurden. Es sind die Werte, die du in der letzten Übung unter „Du bist dran ...“ selbst herausgefunden hast.

- √ Schaue dir noch einmal die Mathematik an, die im Abschnitt Programmierung einer Servosteuerung mit **Potentiometer** auf Seite 164 vorgestellt wurde und verwende deine maximalen und minimalen Werte.
- √ Ersetze die Werte von **scale** und **offset** in ControlServoWithPot.bs2.
- √ Füge die folgende Programmzeile zwischen die Befehle **PULSOUT** und **LOOP** ein, damit die Ergebnisse sichtbar werden.

```
DEBUG HOME, DEC5 time    ' Display adjusted time value.
```

- √ Führe das geänderte Programm aus und überprüfe deine Arbeit. Da die Werte gerundet wurden, liegen die Grenzen in der Nähe von 500 und 1000.

Deklaration von Konstanten in Programmen

In größeren Programmen kommt es häufig vor, dass der Befehl **PULSOUT**, der Skalierungsfaktor (Einstellung hier: 185) und das Offset (Einstellung hier: 500) mehrfach aufgerufen werden. Mit Hilfe der Direktive **CON** lassen sich Werten sogenannte Aliasnamen zuweisen, wie die folgenden Beispiele zeigen:

```
scaleFactor CON 185
offset CON 500
```

Die Aliasnamen werden, wie bei der Festlegung des Variablentyps auch, zu Beginn eines Programms deklariert.

Wird an irgendeiner Stelle innerhalb eines Programms einer dieser Werte benötigt, muss nur der entsprechend deklarierte Begriff **offset** oder **scaleFactor** statt des Zahlenwertes eingegeben werden. Ein Beispiel:

```
time = time */ scaleFactor                    ' Scale by 0.724.
time = time + offset                          ' Offset by 500.
```

Die gleiche Technik lässt sich auch auf die I/O Pins übertragen. Zum Beispiel lässt sich der I/O Pin P7 als Konstante deklarieren.

```
rcPin    CON 7
```


Im letzten Beispielprogramm gab es zwei Stellen, an denen sich die Zahl 7 auf den I/O Pin P7 bezog. Diese Stellen lassen sich jetzt wie folgt umschreiben:

```
HIGH rcPin
```

Die zweite Stelle lautet neu:

```
RCTIME rcPin, 1, time
```

Wird die Schaltung später geändert, muss nur der Wert der Konstanten in der Konstantendeklaration neu gesetzt werden und die beiden Befehle **HIGH** und **RCTIME** werden automatisch aktualisiert. Ebenso einfach lassen sich der Skalierungsfaktor und das Offset neu einstellen, indem die **CON** Direktiven am Programmanfang geändert werden.

Wenn einer Variablen, Konstanten oder einem I/O Pin mit Hilfe von **VAR**, **CON** oder **PIN** ein Name zugewiesen wird, bezeichnet man das auch als **Zuweisung eines Alias**.

Programmbeispiel: ControlServoWithPotUsingConstants.bs2

Dieses Programm macht Gebrauch von Aliases bei fast allen Zahlaufrufen.

- √ Gib das Programm ControlServoWithPotUsingConstants.bs2 und führe es aus.
- √ Beobachte, wie der Servo auf das Potentiometer reagiert und bestätige, dass es sich wie im vorherigen Beispielprogramm (ControlServoWithPot.bs2) verhält.

```
' What's a Microcontroller - ControlServoWithPotUsingConstants.bs2
' Read potentiometer in RC-time circuit using RCTIME command.
' Apply scale factor and offset, then send value to servo.

' {$STAMP BS2}
' {$PBASIC 2.5}
```

```
scaleFactor    CON     185
offset         CON     500
rcPin          CON     7
delay          CON     10
servoPin       CON     14

time           VAR     Word

DO
  HIGH rcPin
  PAUSE delay
  RCTIME rcPin, 1, time
  time = time */ scaleFactor                ' Scale scaleFactor.
  time = time + offset                      ' Offset by offset.
  PULSOUT servoPin, time                    ' Send pulse to servo.
  DEBUG HOME, DEC5 time                     ' Display adjusted time value.
LOOP
```

Du bist dran –Konstante zur Kalibrierung und schnellen Aktualisierung

Werden die I/O Pins gewechselt, die über die Befehle **`HIGH`** und **`RCTIME`** angesprochen werden, muss nur der Wert der Konstanten **`rcPin`** angepasst werden.

- √ Speicher das Beispielprogramm unter einem neuen Namen ab.
- √ Gib als Werte für **`scaleFactor`** und **`offset`** die Werte ein, die sich aus der RC-Schaltung des letzten Abschnitts „Du bist dran ...“ ergaben.
- √ Führe das geänderte Programm aus und überprüfe, dass es korrekt arbeitet.
- √ Änder die Schaltung so ab, dass die RC-Schaltung von I/O Pin P7 an I/O Pin P8 gelegt wird.
- √ Änder die **`rcPin`** Zuweisung wie folgt ab:

```
rcPin    CON 8
```

- √ Füge vor der **`DO…LOOP`** Schleife den folgenden Befehl ein und überzeuge dich, dass die Konstante **`rcPin`** den Wert acht enthält:

```
DEBUG ? rcPin
```

- √ Starte das Programm erneut und überprüfe, ob die Befehle **`HIGH`** und **`RCTIME`** weiterhin korrekt arbeiten.

ZUSAMMENFASSUNG

In diesem Kapitel wurde das Potentiometer eingeführt. Es handelt sich um ein Bauelement, das sich unter vielen Reglern und Einstellknöpfen verbirgt. Es besteht aus einer Widerstandsstrecke, auf die mit einem beweglichen Schleifer zugegriffen werden kann. An den Enden der Widerstandsstrecke und am Schleifer befinden sich Kontakte, über die das Bauteil beschaltet wird. Ein Potentiometer kann als variabler Widerstand eingesetzt werden, indem man den Schleifer und einen Anschlusspunkt der Widerstandsstrecke beschaltet.

In diesem Kapitel wurde auch der Kondensator eingeführt. Er wird dort eingesetzt, wo Ladungen gespeichert oder abgegeben werden sollen. Wieviel Ladung ein Kondensator speichern kann, hängt von dem aufgedruckten Kapazitätswert ab (er wird in Farad (F) angegeben). In der wissenschaftlichen Notation wird die Größe *ein millionstel* mit dem griechischen Buchstaben µ abgekürzt. Die Kapazitäten der in den Übungen dieses Kapitels eingesetzten Kondensatoren liegen zwischen 0.01 und 3300 µF.

Ein Widerstand und ein Kondensator in einem Schaltkreis lassen sich so beschalten, dass sich der Kondensator innerhalb einer bestimmten Zeit auf- und entlädt. Eine solche Schaltung nennt man kurz RC-Schaltung. Die Buchstaben R und C in einem RC-Kreis bezeichnen die Bauteile Widerstand und Kondensator. Wird einer der Werte (C in der Übung dieses Kapitels) konstant gehalten, dann hängt die Entladezeit des Kondensators nur vom Wert von R ab. Ändert sich der Wert des Widerstandes, ändert sich auch die Lade- und Entladezeit. Die komplette Entladezeit einer RC-Schaltung lässt sich über die Größe des Kapazitätswertes beeinflussen.

Mit Hilfe einer Datenabfrage wurde der Entladevorgang eines Kondensators mit großer Kapazität in einer RC Schaltung aufgezeichnet. Mit Widerständen unterschiedlicher Größe wurde gezeigt, wie sich die Entladezeit ändert, wenn sich der Widerstandswert ändert. Mit dem **`RCTIME`** Befehl wurde ein Potentiometer in einer RC-Schaltung mit kleiner Kapazität untersucht. Obwohl die Entladezeiten bei den eingesetzten Kondensatoren zwischen 2 und 1500 µs liegen, gelingt es ohne Schwierigkeiten mit Hilfe der BASIC Stamp die Zeitmessungen mit dem **`RCTIME`** Befehl aufzuzeichnen. Dazu muss der I/O Pin auf **`HIGH`** gesetzt und die Kapazität im RC-Schaltkreis vor Ausführung des Befehls **`RCTIME`** über den **`PAUSE`** Befehl die Möglichkeit erhalten, sich aufladen zu können.

Mit Hilfe der PBASIC Programmierung lassen sich auch Potentiometer ausmessen und in Verbindung mit Servos einsetzen. Über die gemessene Entladezeit einer RC-Schaltung, die der Befehl **RCTIME** in einer Variablen ablegt, lassen sich weitere mathematische Berechnungen durchführen. Die Variable kann durch Addition einer Konstanten justiert werden, was sich bei der Steuerung von Servos gut einsetzen lässt. Im Abschnitt Projekte wurde auf die Multiplikation und Division eingegegangen. Die Direktive **CON** kann zu Beginn eines Programmes eingesetzt werden, um einer konstanten Zahl einen Namen zuzuordnen. Wie bei der Festlegung einer Variablen wird die Festlegung von Konstanten als Alias-Zuweisung bezeichnet. Wurde ein Alias erzeugt, dann lässt sich der Name, statt der Zahl, im kompletten Programm nutzen. Geschieht dies an vielen verschiedenen Stellen innerhalb eines Programms, dann wird der Vorteil dieser Schreibweise sofort deutlich, wenn der Zahlenwert durch einen anderen ausgetauscht werden muss. Statt an allen Programmstellen neue Zahlenwerte einzugeben, genügt es, über die Direktive **CON** einen neuen Zahlenwert zuzuweisen und das Programm neu zu starten.

Verständnisfragen

1. Welches elektronische Bauteil verbirgt sich häufig hinter dem Einstellknopf einer Audioanlage?
2. Ist bei einem Potentiometer der Widerstand zwischen den beiden äußeren Anschlusspunkten veränderbar?
3. Welche Gemeinsamkeiten gibt es zwischen einer Kapazität und einer wiederaufladbaren Batterie?
4. Wie kann man bei einer RC-Schaltung den Widerstandswert des Potentiometers bestimmen?
5. Wie ändert sich in einer RC-Schaltung die Entladezeit, wenn der Widerstand R größer/kleiner wird?
6. Was macht die **CON** Direktive? Erkläre dies an einem praktischen Beispiel.

Übung

1. Du hast eine 0.5 µF Kapazität in einer RC Schaltung, und möchtest, dass die Messdauer sich verzehnfacht. Welchen Wert muss der neue Kondensator haben?

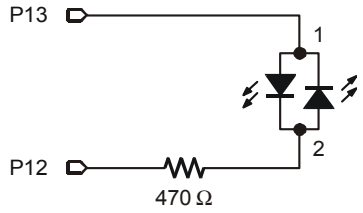
Projekte

1. Füge zur Übung 4 eine DUO-LED Schaltung hinzu. Veränder das Beispiel-programm so, dass die DUO-LED rot leuchtet, wenn der Servo in entgegengesetzter Uhrzeigerrichtung, grün, wenn er in Uhrzeigerrichtung rotiert und die LED erlischt, wenn der Servo seine Position hält.

2. Veränder mit der Anweisung **IF…THEN** das Beispielprogramm aus Übung 4 so, dass der Servo nur zwischen den **PULSOUT** Werten von 650 und 850 rotiert.

Lösungen

Q1. Ein Potentiometer.
Q2. Nein, er ist fest. Der variable Widerstand liegt zwischen einem der beiden äußeren Anschlusspunkte und dem Mittelabgriff..
Q3. Ein Kondensator verhält sich wie eine wiederaufladbare Batterie, da er wie diese aufgeladen werden kann und damit eine Spannung an seinen Polen anliegt. Der einzige Unterschied besteht darin, dass er die Ladungen nur sehr kurz halten kann.
Q4. Die Entladezeit (oder Ladezeit) eines Kondensator lässt sich messen. Diese Zeit hängt vom Widerstandswert und der Kapazität ab. Ist die Kapazität bekannt und der Widerstand variabel, dann gibt die Entladezeit einen Hinweis auf den Widerstand.
Q5. Wird R größer, wächst die Entladezeit direkt proportional zu R. Wird R kleiner, fällt die Entladezeit direkt proportional zu R.
Q6. Die Direktive **CON** ersetzt eine Zahl durch einen Namen.
E1. new cap = 10 x old cap value
= 10 x 0.5µF
= 5µF
P1. Übung 4 mit DUO-LED.



Potentiometerschaltung aus Abbildung 5-11 S. 159, Servo aus Abbildung 4-3 S. 112, DUO-LED aus Abbildung 2-19 S. 65, mit P15 und P14 geändert in P13 und P12.

```
' What's a Microcontroller - Ch5Prj01_ControlServoWithPot.bs2
' Read potentiometer in RC-time circuit using RCTIME command.
' The time variable ranges from 126 to 713, and an offset of 330 is
' needed.
' Bi-color LED on P12, P13 tells direction of servo rotation:
' green for CW, red for CCW, off when servo is holding position.

' {$STAMP BS2}
' {$PBASIC 2.5}
DEBUG "Program Running!"
```

```
time           VAR     Word            ' time reading from
potentiometer
prevTime       VAR     Word            ' previous reading

DO
  prevTime = time                      ' Store previous time reading
  HIGH 7                               ' Read pot using RCTIME
  PAUSE 10
  RCTIME 7, 1, time
  time = time + 330                    ' Scale pot, match servo range

  IF ( time > prevTime + 2) THEN       ' increased, pot turned CCW
    HIGH 13                            ' Bi-color LED red
    LOW 12
  ELSEIF ( time < prevTime - 2) THEN   ' value decreased, pot turned CW
    LOW 13                             ' Bi-color LED green
    HIGH 12
  ELSE                                 ' Servo holding position
    LOW 13                             ' LED off
    LOW 12
  ENDIF

  PULSOUT 14, time
LOOP
```

P2. Der Schlüssel sind die hinzugefügten **`IF…THEN`** Blöcke; ein Beispiel zeigt:

```
' What's a Microcontroller - Ch5Prj02_ControlServoWithPot.bs2
' Read potentiometer in RC-time circuit using RCTIME command.
' The time variable ranges from 126 to 713, and an offset of 330 is
' needed.
' Modify so the servo only rotates from 650 to 850.
' {$STAMP BS2}
' {$PBASIC 2.5}
DEBUG "Program Running!"

time VAR Word

DO
  HIGH 7                               ' Read pot with RCTIME
  PAUSE 10
  RCTIME 7, 1, time

  time = time + 330                    ' Scale time to servo range

  IF (time < 650) THEN                 ' Constrain range from 650 to 850
    time = 650
  ENDIF
```

```
  IF (time > 850) THEN
    time = 850
  ENDIF

  PULSOUT 14, time
LOOP
```

Weiterführende Bücher

Verschiedene unterschiedliche elektronische Bauelemente, Konzepte und Techniken wurden in diesem Kapitel aufgezeigt. Einige der bedeutenderen Beispiele sind:

- Ein Potentiometer als Eingabegerät
- Messung von Widerstand/Kapazität eines Bauteils mit Hilfe von **RCTIME**
- Eingabewerte mathematisch verändern und wieder ausgeben
- Einen Motor über Messwerte steuern

"*Advanced Robotics: with the Toddler*", Student Workbook, Version 1.2, Parallax Inc., 2003
"*Robotics with the Boe-Bot*", Student Workbook, Version 2.0, Parallax Inc., 2003
"*SumoBot*", Student Workbook, Version 1.1, Parallax Inc., 2002

Jedes Stamps in Class Robotics Buch wertet mit dem **RCTIME** Befehl die unterschiedlichen Sensoren aus und erhält eine Vielzahl von Reaktionsmustern. Jedes Reaktionsmuster führt mit Rechenaufwand zu einer Entscheidung, die mit einer Bewegung des Roboters abgeschlossen wird.

"*Basic Analog and Digital*", Student Guide, Version 2.0, Parallax Inc., 2003

Basic A*nalog and Digital* erzeugt mit einem Potentiometer veränderliche Spannungswerte, die mit einem analog-digital Konverter ausgewertet werden. Mit einem Potentiometer als Eingabegerät wird die Frequenz eines 555-Timer Bausteines gesetzt. In diesem Buch wird auch auf die Mathematik des Spannungsabfalls in RC-Schaltungen näher eingegangen.

"*Applied Sensors*", Student Guide, Version 1.3, Parallax Inc., 2003

Mit dem intensiven Gebrauch des **RCTIME** Befehl werden in diesem Buch Daten verschiedener Sensoren aufgenommen.

"*Industrial Control*", Student Guide, Version 2.0, Parallax Inc., 2002

Dieses Buch gibt eine Einführung in industrielle Techniken, mit denen Maschinen heute über Sensoreingaben überwacht und gesteuert werden. Die Techniken beschränken sich auf Steuerungssysteme.

Kapitel 6: Displays ansteuern

DISPLAYS INFORMIEREN

Abbildung 6-1 zeigt das Display einer Mikrowelle. Im nicht benutzten Zustand erscheint dort die aktuelle Uhrzeit. Wird eine Mikrowelle aktiviert, zeigt es die Timerzeit mit der gewählten Programmeinstellung. Ist die Garzeit abgelaufen, beginnt es zu blinken, sobald der Alarm ertönt. Ein Mikrocontroller in der Gerätetür überwacht die Tastenfelder und aktualisiert die Anzeige. Ebenso überwacht er Sensoren im Gerät und bedient Schalter, die das Heizelement ein- und ausschalten.

Abbildung 6-1
Digitale 7-Segment Uhranzeige in einer Mikrowelle

Jede der drei Ziffern in Abbildung 6-1 wird auf einer sogenannten 7-Segment-Anzeige dargestellt. In diesem Kapitel werden wir die BASIC Stamp so programmieren, dass Zahlen und Buchstaben auf einer 7-Segment-Anzeige dargestellt werden.

AUFBAU EINER 7-SEGMENT-ANZEIGE

Eine 7-Segment-Anzeige ist ein quaderförmiger Block mit 7 geometrisch angeordneten gleichlangen Leuchtdioden, die einzeln zum Leuchten gebracht werden können. Abbildung 6-2 zeigt eine Bauteildarstellung einer sehr häufig benutzten 7-Segment-LED Anzeige, die wir in den Übungen dieses Kapitels einsetzen werden. Eine zusätzliche LED, kann als Dezimalpunkt ebenfalls genutzt werden. Jedes Segment (A bis G) und der Punkt repräsentieren jeweils eine LED, die einzeln angesteuert werden können. Die meisten Anschlusspins sind mit einer Zahl und einem Buchstaben, der für ein LED-Segment steht, gekennzeichnet. Pin 5 (DP) bezeichnet den Dezimalpunkt. Pin 3 und 8 (“common cathode”) wird erklärt, sobald wir uns die Schaltung dieses Bauteils anschauen.

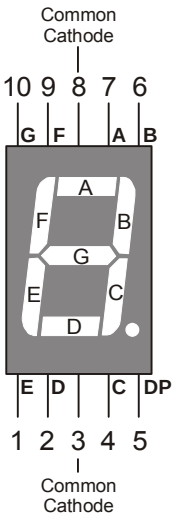


Abbildung 6-2
7-Segment LED
Anzeige
Bauteildarstellung
und Pinbelegung

Pinbelegung: Abbildung 6-2 ist ein Beispiel für eine Pinbelegung. Sie gibt nützliche Informationen zur Beschaltung des Bauteils. Pinbelegungen weisen jedem Pin eine Zahl, einen Buchstaben und einen Bezug zu; siehe Abbildung 6-2. Jeder Pin ist nummeriert und korrespondiert mit einer bestimmten Segmentbezeichnung. Der Bezug wird durch die Draufsicht auf das Bauteil hergestellt. Aus ihr ergibt sich, dass Pin 1 unten links am Display zu finden ist. Andere Bauteile weisen raffiniertere Bezüge aus, wie zum Beispiel die Abflachung an einer LED.

Abbildung 6-3 zeigt den inneren Schaltungsaufbau einer 7-Segment LED Anzeige. Jede Anode der LEDs ist mit einem separatem Pin verbunden, während die Kathoden miteinander verbunden sind (gemeinsame Kathode). Aus diesem Grund wird eine solche 7-Segment-Anzeige auch als Anzeige mit „gemeinsamer Kathode“ bezeichnet. Wird Pin 3 oder Pin 8 mit Vss verbunden, sind alle LED-Kathoden mit Vss verbunden.

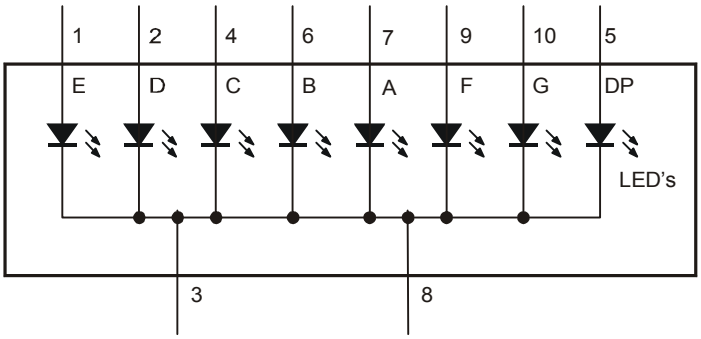


Abbildung 6-3
Schaltungsaufbau 7-Segment Anzeige

ÜBUNG 1: AUFBAU UND TESTSCHALTUNG MIT EINER 7-SEGMENT ANZEIGE

In dieser Übung werden Schaltungen aufgebaut, mit denen sich die Segmente einer Anzeige testen lassen.

Schaltungsbauteile für 7-Segment Anzeige

1 7-Segment LED Anzeige
5 Widerstände – 1 kΩ (braun-schwarz-rot)
5 Verbindungsdrähte

Testschaltung: 7-Segment Anzeige

√ Trenne die Spannungsversorgung vom Board of Education oder HomeWork Board und baue die Schaltung aus Abbildung 6-4 und Abbildung 6-5 auf.
√ Schalte die Spannungsversorgung wieder an und überzeuge dich, dass das Segment A aufleuchtet.

Welche Bedeutung hat das x über dem nc in der Schaltung? Das nc steht für "not connected" oder "no-connect". Es zeigt an, dass einzelne Pins der 7-Segment-LED-Anzeige keine Belegung haben. Das x am Ende eines Pins weist auf keine Belegung hin. In Schaltungen wird manchmal nur das x-Symbol oder der nc-Hinweis gegeben.

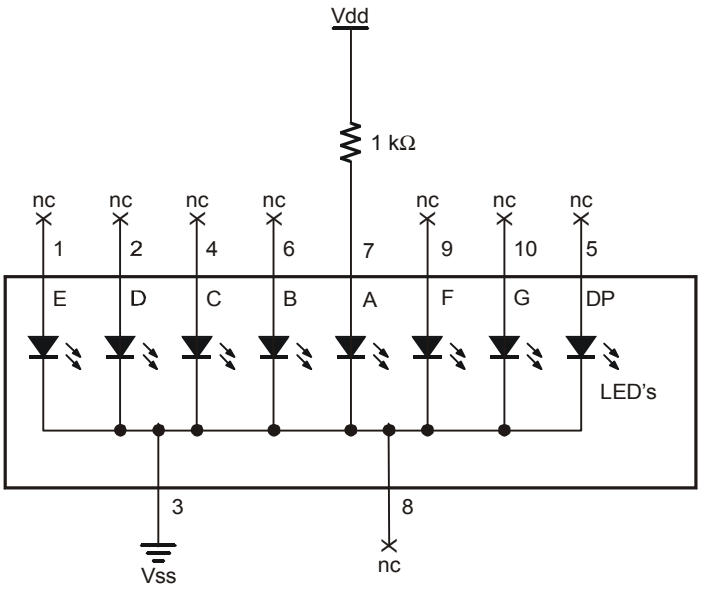


Abbildung 6-4
Testschaltung für die Anzeige des 'A' Segments auf einer 7-Segment LED Anzeige.

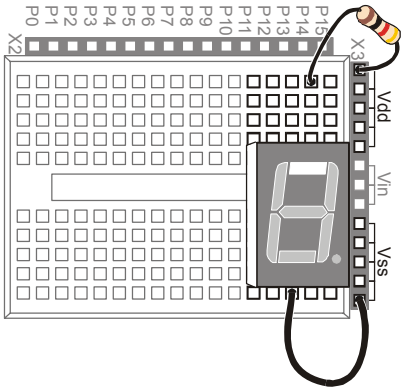


Abbildung 6-5
Testschaltung Verdrahtungsplan für die Anzeige des 'A'-Segments auf einer 7-Segment-LED Anzeige

√ Trenne die Spannungsversorgung und veränder die Schaltung so, dass der Widerstand mit der B LED verbunden wird, wie in Abbildung 6-6 und Abbildung 6-7 gezeigt.

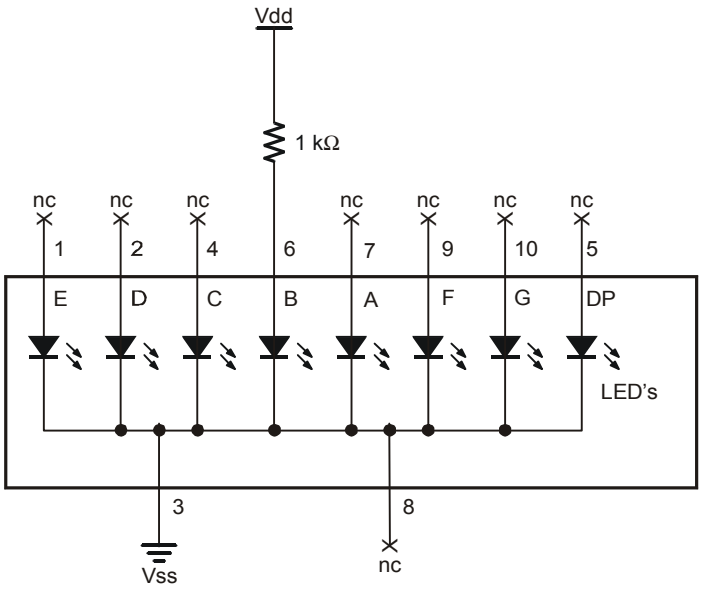


Abbildung 6-6
Testschaltung für die Anzeige des 'B' Segments einer 7-Segment LED Anzeige.

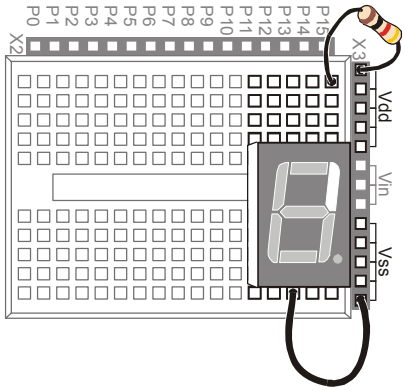


Abbildung 6-7
Testschaltung Verdrahtungsplan für die Anzeige des 'B'-Segments auf einer 7-Segment-LED Anzeige

√ Lege die Spannung wieder an und überprüfe, ob das B-Segment aufleuchtet.
√ Benutze die Pinbelegung aus Abbildung 6-2 als Hilfe, um die anderen Segmente C bis G zum Leuchten zu bringen.

Du bist dran – Der Buchstabe A und die Zahl 2

Abbildung 6-8 und Abbildung 6-9 zeigen wie die Zahl ‘3’ in einer 7-Segment-LED Anzeige verdrahtet werden muss.

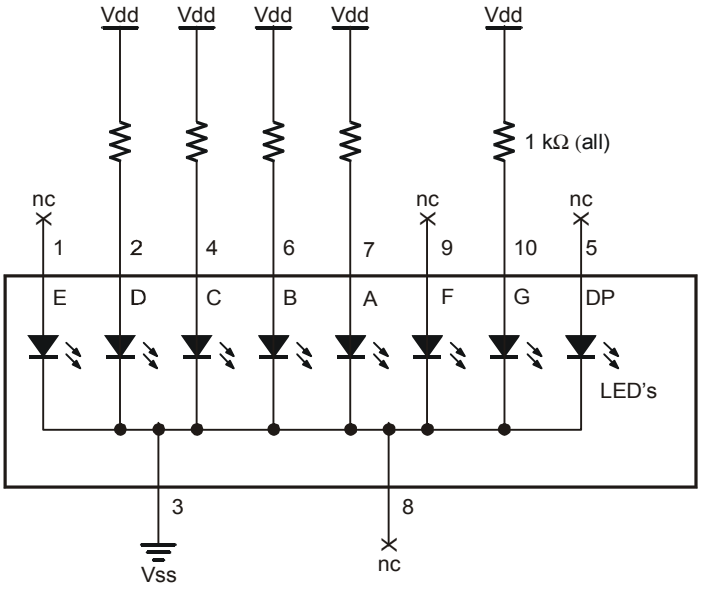


Abbildung 6-8
Verdrahtungsplan für die Zahl ‘3’

Mit Hilfe dieser Verdrahtung wird die Zahl “3” auf der 7-Segment-LED Anzeige angezeigt.

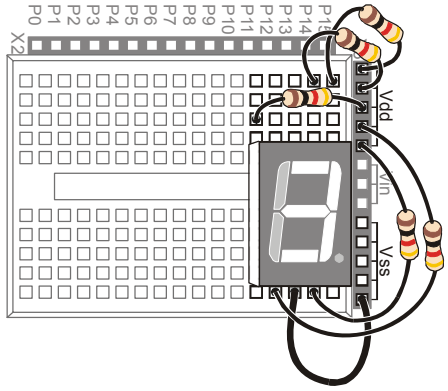


Abbildung 6-9
Verdrahtungsplan für Abbildung 6-8

√ Baue und teste die Schaltung aus Abbildung 6-8 und Abbildung 6-9. Überprüfe, ob die Zahl 3 dargestellt wird.

√ Gib den Schaltungsplan an, mit dem sich die Zahl 2 auf einer 7-Segment-LED Anzeige darstellen lässt.
√ Baue und teste die Schaltung und überprüfe, ob sie wie geplant arbeitet.
√ Probiere auch den Buchstaben ‘A’ darzustellen.

ÜBUNG 2: ANSTEUERUNG EINER 7-SEGMENT ANZEIGE

In dieser Übung wird die 7-Segment Anzeige mit der BASIC Stamp verbunden und ein einfaches Programm ausgeführt, das jedes LED-Segment überprüft.

Bauteile für 7-Segment Anzeige

1 7-Segment-LED Anzeige
8 Widerstände – 1 kΩ (braun-schwarz-rot)
5 Verbindungsdrähte

7-Segment Anzeige mit BASIC Stamp verbinden

Abbildung 6-10 zeigt die Schaltung, Abbildung 6-11 den Verdrahtungsplan für die BASIC Stamp gesteuerte 7-Segment-LED Anzeige.

√ Baue die Schaltung nach Abbildung 6-10 und Abbildung 6-11 auf.

Schaltung und Pinbelegung: Wenn man versucht, die Schaltung nur nach dem Schaltbild in Abbildung 6-10 aufzubauen, ohne Abbildung 6-11 zu benutzen, muss die Pinbelegung der 7-Segment-LED Anzeige zu Hilfe genommen werden(Abbildung 6-2, Seite 176).

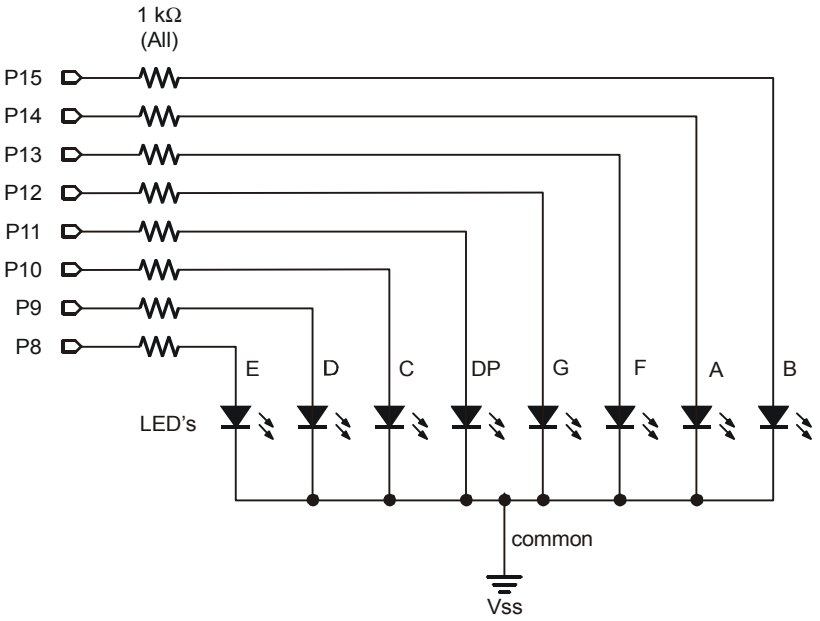


Abbildung 6-10
Schaltung einer BASIC Stamp gesteuerten 7-Segment-LED Anzeige

Achtung bei der Verbindung der Widerstände mit P13 und P14. Achte sehr genau auf die Widerstände, die mit P13 und P14 in Abbildung 6-11 verbunden sind. Es ist eine Freistelle zwischen diesen beiden Widerständen, die sich dadurch ergibt, dass Pin 8 der 7-Segment-LED Anzeige nicht verbunden wird. Ein Widerstand verbindet I/O Pin P13 mit der 7-Segment-LED Anzeige Pin 9 und ein weiterer Widerstand P14 mit dem Anzeigepin 7.

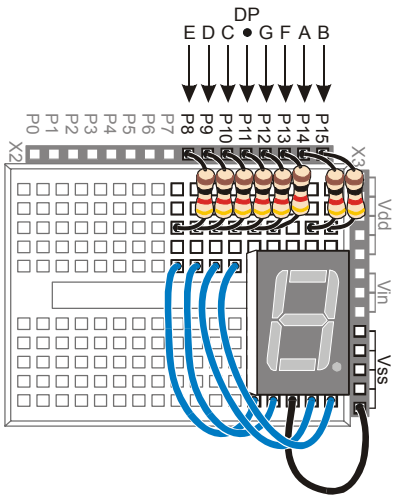


Abbildung 6-11
Verdrahtungsplan für Abbildung 6-10

Benutze die Segment-bezeichnungen oberhalb des Boards als Bezugs-größen.

Parallel Device: Eine 7-Segment-LED Anzeige wird auch als Parallel Device bezeichnet, da zur Ansteuerung mehr als eine I/O Leitung notwendig ist. Im vorgestellten Fall werden 8 I/O Leitungen zur Ansteuerung benötigt.

Parallelbus: Die Übertragungsstrecken für die einzelnen **`HIGH/LOW`** Signale von der BASIC Stamp zur 7-Segment-LED Anzeige werden als Parallelbus bezeichnet. In Abbildung 6-10 werden sie als parallele Linien gezeichnet.

Programmierung einer 7-Segment Displayschaltung

Die Befehle **`HIGH`** und **`LOW`** werden mit einer Variablen als Pinargument aufgerufen. Um jedes Segment einzeln austesten zu können, werden die beiden Befehle in eine **`FOR…NEXT`** Schleife eingebunden und über den Laufindex wird anschließend der I/O Pin auf HIGH und LOW gesetzt.

- √ Starte das Programm SegmentTestWithHighLow.bs2.
- √ Überprüfe, ob jedes Segment in der 7-Segment-LED Anzeige kurz aufleuchtet und wieder ausgeht.
- √ Lege eine Liste an, in der dargestellt ist, welches Segment durch welchen I/O Pin angesteuert wird.

Programmbeispiel: SegmentTestWithHighLow.bs2

```
' What's a Microcontroller - SegmentTestWithHighLow.bs2
' Individually test each segment in a 7-Segment LED display.

'{$STAMP BS2}
'{$PBASIC 2.5}

pinCounter     VAR     Nib

DEBUG "I/O Pin", CR,
      "-------", CR

FOR pinCounter = 8 TO 15

  DEBUG DEC2 pinCounter, CR
  HIGH pinCounter
  PAUSE 1000
  LOW pinCounter

NEXT
```

Du bist dran – Ein anderes Muster

Es gibt einen überraschenden Effekt, wenn der Befehl **`LOW pinCounter`** gelöscht wird:

- √ Kommentiere den Befehl **`LOW pinCounter`** mit einem Apostroph am Zeilenanfang aus.
- √ Führe das geänderte Programm aus und beobachte die Reaktion.

ÜBUNG 3: ZIFFERN AUF DISPLAY AUSGEBEN

Einschließlich Dezimalpunkt benötigt die BASIC Stamp acht I/O Pins zur Ansteuerung der 7-Segment Anzeige. Machen wir uns das noch einmal klar: mit acht **`HIGH`** oder **`LOW`** Signalen wird eine Zahl auf dem Display dargestellt. Der einfache Zählvorgang von 1 bis 9 verlangt schon einen erheblichen Programmieraufwand. Über spezielle Variablen lassen sich HIGH und LOW Werte für mehrere I/O Pins gleichzeitig setzen.

In dieser Übung werden über 8-stellige Binärzahlen und nicht über die Befehle **`HIGH`** und **`LOW,`** die an den I/O Pins der BASIC Stamp anliegenden HIGH/LOW Signale gesteuert. Mit nur einem PBASIC Befehl lassen sich über die Variablen **`DIRH`** und **`OUTH`** die HIGH/LOW Signale aller mit einer 7-Segment Anzeige verbundenen I/O Pins gleichzeitig ansprechen.

8 Bit: Eine 8-stellige Binärzahl setzt sich aus 8 einzelnen Bits zusammen. Jedem Bit entspricht der Zustand 1 oder 0.

Ein Byte besteht aus einer Zeichenfolge von 8 Bit. Im Binärsystem lassen sich mit 8-stelligen Zahlen 256 unterschiedliche Zustände darstellen. Eine Variable vom Typ Byte kann damit eine Zahl zwischen 0 und 255 darstellen.

Bauteile und Schaltung für die Darstellung von Ziffern

Siehe vorherige Übung

Programmierung von Ein/Aus Mustern mit Binärzahlen

In dieser Übung experimentieren wir mit den Variablen **DIRH** und **OUTH. DIRH** steuert die Signalrichtung (Eingabe oder Ausgabe) der I/O Pins P8 bis P15. **OUTH** setzt die HIGH oder LOW Zustände der acht I/O Pins. Dieser Befehl ist deshalb so hilfreich, weil mit ihm acht verschiedene I/O Pins zur gleichen Zeit mit einem einzigen Befehl auf HIGH oder LOW gesetzt werden können. Im folgenden Beispielprogramm wird gezeigt, wie mit diesen beiden Variablen ein Zählvorgang von 0 bis 9 auf einer 7-Segment Anzeige dargestellt werden kann, ohne dass die Befehle **HIGH** und **LOW** benutzt werden:

Programmbeispiel: DisplayDigits.bs2

Bei diesem Programm durchläuft die 7-Segment LED Anzeige die Zahlen von 0 bis 9.

- √ Gib das Programm DisplayDigits.bs2 ein und führe es aus.
- √ Überzeuge dich, dass die Zahlen von 0 bis 9 nacheinander angezeigt weden.

```
' What's a Microcontroller - DisplayDigits.bs2
' Display the digits 0 through 9 on a 7-segment LED display.

'{$STAMP BS2}
'{$PBASIC 2.5}

DEBUG "Program Running!"

OUTH = %00000000                            ' OUTH initialized to low.
DIRH = %11111111                            ' Set P8-P15 to all output-low.
                                            ' Digit:
'       BAFG.CDE
OUTH = %11100111                            ' 0
PAUSE 1000
OUTH = %10000100                            ' 1
PAUSE 1000
OUTH = %11010011                            ' 2
```

```
PAUSE 1000
OUTH = %11010110                        ' 3
PAUSE 1000
OUTH = %10110100                        ' 4
PAUSE 1000
OUTH = %01110110                        ' 5
PAUSE 1000
OUTH = %01110111                        ' 6
PAUSE 1000
OUTH = %11000100                        ' 7
PAUSE 1000
OUTH = %11110111                        ' 8
PAUSE 1000
OUTH = %11110110                        ' 9
PAUSE 1000

DIRH = %00000000                        ' I/O pins to input,
                                        ' segments off.
END
```

Arbeitsweise von DisplayDigits.bs2

Abbildung 6-12 zeigt, wie man mit den Variablen **DIRH** und **OUTH** die Ausgaberichtung und den Zustand (HIGH/LOW) der I/O PINS P8 bis P15 steuert.

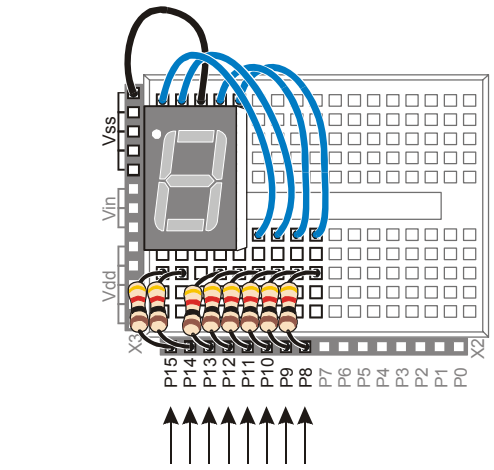


Abbildung 6-12
Benutze DIRH und OUTH und setze alle I/O Pins auf Ausgabe-Low

```
OUTH = %00000000
DIRH = %11111111
```

Der erste Befehl:

```
OUTH = %00000000
```

setzt alle I/O Pins (P8 bis P15) in den Zustand LOW. Wird die Datenrichtung auf Ausgabe geschaltet, erlöscht die 7-Segment Anzeige, alle Segmente werden auf Null gesetzt. **OUTH = %11111111** setzt alle I/O Pins auf HIGH, alle Segmente der 7-Segment LED Anzeige leuchten.

Welche Bedeutung hat %? Das Zeichen % vor einer Ziffernfolge signalisiert dem BASIC Stamp Editor, dass die nachfolgende Zahl eine Binärzahl ist. Der binären Zahl %00001100 entspricht zum Beispiel die Dezimalzahl 12. In dieser Übung wirst du erkennen, wie mit Hilfe binärer Zahlen viele Programmieraufgaben leichter umsetzbar sind.

Die LOW-Signalzustände werden solange nicht an die I/O Pins gelegt, bis über die Variable **DIRH** die Signalrichtungen aller I/O Pins von Eingang auf Ausgang umgeschaltet wurden. Der dazu notwendige Befehl lautet:

```
DIRH = %11111111
```

Da der Befehl **OUTH = %00000000** vor dem Befehl **DIRH** ausgeführt wurde, liegt das LOW-Signal nach Programmausführung an P8 bis P15 an. Mit **DIRH = %00000000** werden alle I/O Pins wieder als Eingänge deklariert.

Bevor aus I/O Pins Ausgänge werden: Solange die Signalrichtung der I/O Pins nicht von Eingang auf Ausgang geändert wurde, werden die an den Pins anliegenden Eingagnssignale in der Variablen **INH** ständig aktualisiert. Sie umfasst **IN8**, **IN9** bis einschließlich **IN15**. Diese Variablen lassen sich in ähnlicher Weise nutzen, wie **IN3** und **IN4** beim Lesen von Tastern in Kapitel 3.

Alle BASIC Stamp I/O Pins sind standardmäßig als Eingänge geschaltet. Das ist die „Voreinstellung". Ein BASIC Stamp I/O Pin muss zunächst als Ausgang definiert werden, bevor man ein HIGH oder LOW Signal an ihn übertragen kann. Die Befehle **HIGH** und **LOW** setzen automatisch die Datenrichtung eines BASIC Stamp I/O Pins auf Ausgang. Eine 1 in der Variablen **DIRH** setzt einen der I/O Pins auf Ausgang.

Abbildung 6-13 zeigt, wie die Variable **OUTH** aufzurufen ist, um ein differenziertes Signal aus Nullen und Einsen an P8 bis P15 zu senden. Eine binäre 1 entspricht einem HIGH-Signal, eine binäre 0 entsprechend einem LOW-Signal. Das folgende Beispiel stellt die Zahl drei auf der 7-Segment Anzeige dar:

```
'       BAFG.CDE
```

```
OUTH = %11010110
```

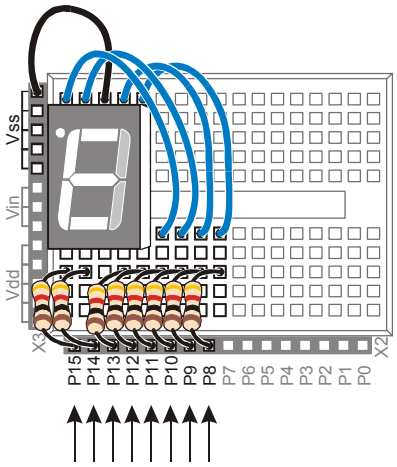


Abbildung 6-13
Mit OUTH werden die HIGH/LOW Signale von P8 bis P15 gesteuert.

```
'          BAFG.CDE
OUTH = %11010110
```

Die Zahl drei wird auf der um 90° gedrehten Anzeige seitenverkehrt angezeigt, da es in dieser Darstellung einfacher zu erklären ist, wie die Werte in **OUTH** mit den I/O Pins zusammenhängen. In dem Befehl **OUTH = %11010110** werden mit den binären Nullen die I/O Pins P8, P11 und P13 auf LOW und mit einer binären Eins die I/O Pins P9, P10, P12, P14 und P15 auf HIGH gesetzt. In der Kommentarzeile vor der Befehlszeile ist dargestellt, welches Segment welchem Stellenwert der Binärzahl zugeordnet wird und ob es ein- oder ausgeschaltet ist. Das folgende Beispielprogramm zeigt, wie die Variable **OUTH** einzustellen ist, um auf der 7-Segment Anzeige einen Zählvorgang von 0 bis 9 ablaufen zu lassen.

Die Befehle HIGH und LOW: Der Befehl **HIGH 15** entspricht der Befehlsfolge **OUT15 = 1** und **DIR15 = 1**. Ebenso entspricht der Befehl **LOW 15** der Befehlsfolge **OUT15 = 1** und **DIR15 = 1**. Soll P15 wieder als Eingang geschaltet werden, setze **DIR15 = 0**. Benutze dann **IN15** (statt SEND) um HIGH/LOW Signale wahrzunehmen.

Du bist dran – Darstellung von A bis F

√ Bestimme zunächst die Bitmuster (Kombination aus Nullen und Einsen) der Buchstaben A, b, C, d, E und F.

√ Veränder SevenSegment0to9 so, dass die Buchstaben A, b, C, d, E, F angezeigt werden..

Dezimal vs. Hexadezimal Die 10 Ziffern des Zehnersystems (10-er Basis) sind: 0, 1, 2, 3, 4, 5, 6, 7, 8, und 9. Im Hexadezimalsystem (16-er Basis) lauten die 16 Ziffern: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Die 16-er Basis wird intensiv sowohl in der Computer- als auch der Mikrocontrollerprogrammierung eingesetzt. Sobald du herausgefunden hast, wie die Buchstaben A bis F dargestellt werden, änder das Programm so ab, dass die 16 Ziffern des Hexadezimalsystems nacheinander erscheinen.

Listen von Ein/Aus Mustern

Der Befehl **LOOKUP** gestaltet das Schreiben von Code für 7-Segment LED Anzeigemustern zukünftig wesentlich leichter. Mit ihm lassen sich einzelne Elemente in einer Liste nachschlagen. Es folgt ein Codierungsbeispiel:

```
LOOKUP index, [7, 85, 19, 167, 28], value
```

Der Befehl enthält zwei Variable, **index** und **value**. Wenn **index** identisch 0 ist, speichert **value** die **7**. Wenn **index 1** ist, speichert **value 85**. Da in diesem Beispiel der **index** identisch 2 ist, weist der **LOOKUP** Befehl der Variablen **value** den Wert **19** zu und das Debug Terminal zeigt diesen Wert an.

Programmbeispiel: SimpleLookup.bs2

√ Führe das Programm SimpleLookup.bs2 aus.
√ Führe das Programm wie gehabt aus, setze aber die Variable **index** auf 2.
√ Versuche die Variable **index** auf eine Zahl zwischen 0 und 4 zu setzen.
√ Führe das Programm nach jeder Änderung der Variablen **index** neu aus und beobachte, welcher Wert aus der Liste der Variablen **value** zugeordnet wird.
√ Optional: Änder das Programm so ab, dass der Befehl **LOOKUP** in einer **FOR…NEXT** Schleife, die von 0 bis 4 durchlaufen wird, eingebettet ist.

```
' What's a Microcontroller - SimpleLookup.bs2
' Debug a value using an index and a lookup table.

' {$STAMP BS2}
' {$PBASIC 2.5}

value          VAR     Byte
index          VAR     Nib

index = 2
```

```
DEBUG ? index

LOOKUP index, [7, 85, 19, 167, 28], value

DEBUG ? value, CR

DEBUG "Change the index variable to a ", CR,
      "different number(between 0 and 4).", CR, CR,

      "Run the modified program and ", CR,
      "check to see what number the", CR,
      "LOOKUP command places in the", CR,
      "value variable."

END
```

Programmbeispiel: DisplayDigitsWithLookup.bs2

Dieses Beispielprogramm zeigt, wie sich mit Hilfe des **LOOKUP** Befehls Bitmuster abspeichern lassen, die später über die Variable **OUTH** aufgerufen werden. Erinnern wir uns, mit dem **index** wird festgelegt, welcher binäre Wert in der Variablen **OUTH** abgelegt wird. Dieses Beispielprogramm wird wie schon das vorherige die Zahlen von 0 bis 9 anzeigen. Der Unterschied zwischen diesem Programm und DisplayDigits.bs2 besteht in der größeren Vielseitigkeit. Umstellungen auf andere Zahlenausgaben gestalten sich über Lookup-Listen wesentlich schneller und leichter.

- √ Führe das Programm DisplayDigitsWithLookup.bs2 aus.
- √ Überprüfe, ob es ebenso arbeitet wie das vorherige Programm (bei geringerem Arbeitsaufwand).
- √ Schau dir das Debug Terminal an, während das Programm läuft. Es demonstriert, wie der Wert von **index** vom **LOOKUP** Befehl umgesetzt wird und der korrekte Binärwert aus der Liste an **OUTH** übergeben wird.

```
' What's a Microcontroller - DisplayDigitsWithLookup.bs2
' Use a lookup table to store and display digits with a 7-segment LED display.

'{$STAMP BS2}
'{$PBASIC 2.5}

index          VAR     Nib

OUTH = %00000000
DIRH = %11111111
```

```
DEBUG "index  OUTH   ", CR,
      "-----  --------", CR

FOR index = 0 TO 9

  LOOKUP index, [ %11100111, %10000100, %11010011,
                  %11010110, %10110100, %01110110,
                  %01110111, %11000100, %11110111, %11110110 ], OUTH

  DEBUG "   ", DEC2 index, "  ", BIN8 OUTH, CR

  PAUSE 1000

NEXT

DIRH = %00000000

END
```


Du bist dran – Hexadezimalzahlen

√ Änder das Programm DisplayDigitsWithLookup.bs2 so ab, dass es von 0 bis F hexadezimal zählt. Vergiss nicht die Werte von **`start`** und **`end`** in der **`FOR…NEXT`** Schleife anzupassen.

ÜBUNG 4: EINE DREHBEWEGUNG AUF DISPLAY ANZEIGEN

In Kapitel 5, Übung 4 haben wir mit Hilfe eines Potentiometers die Position eines Servos gesteuert. In dieser Übung wird die Position eines Potentiometers auf einer 7-Segment Anzeige dargestellt.

Bauteile für Anzeige und Poti

1 7-Segment LED Anzeige
8 Widerstände – 1 kΩ (braun-schwarz-rot)
1 Potentiometer – 10 kΩ
1 Widerstand – 220 Ω (rot-rot-braun)
1 Kondensator – 0.1 µF
7 Verbindungsdrähte

Schaltungsaufbau von Anzeige und Poti

Abbildung 6-14 zeigt den Aufbau einer Potentiometerschaltung, die dem Projekt hinzugefügt wird. Abbildung 6-15 zeigt die Verdrahtung der Schaltung aus Abbildung 6-14 zusammen mit der Schaltung aus Abbildung 6-10 von Seite 182.

√ Verbinde die Potentiometerschaltung mit der 7-Segment Anzeige wie in Abbildung 6-15 dargestellt.

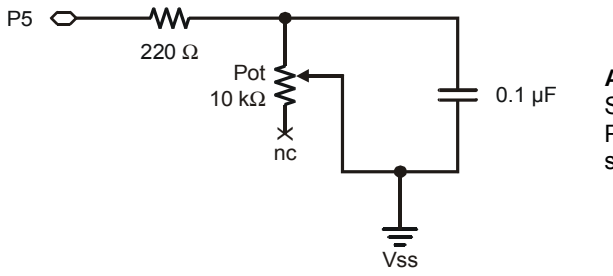


Abbildung 6-14
Schaltung des Potentiometer-schaltkreises

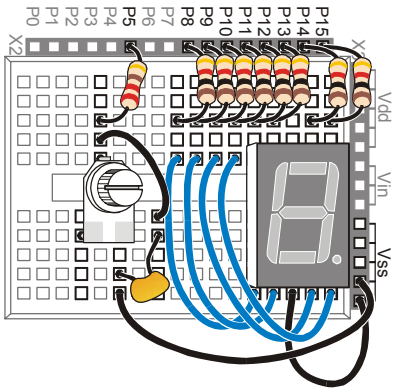


Abbildung 6-15
Verdrahtungsplan des Projekts

Programmierung von Anzeige und Poti

Das Gegenstück zum `LOOKUP` Befehl ist der Befehl `LOOKDOWN`. Während über den `LOOKUP` Befehl eine Zahl über einen Index herausgefunden wird, gibt der `LOOKDOWN` Befehl nach Eingabe einer Zahl einen Index zurück.

Programmbeispiel: SimpleLookdown.bs2

Dieses Beispielprogramm zeigt, wie der **LOOKDOWN** Befehl aufgerufen wird.

- √ Führe das folgende Programm aus: SimpleLookdown.bs2.
- √ Starte das Programm und setze die Variable **value** auf 167. Beobachte über das Debug Terminal den Wert von **index**.
- √ Setze die Variable **value** im Befehl **LOOKDOWN** nacheinander auf folgende Werte: 7, 85, 19, 28.
- √ Starte das Programm nach jeder Änderung von **value** erneut und beobachte, welcher Wert der Liste in die Variable **index** eingesetzt wird.

Mit Hilfe des **LOOKDOWN** Befehls lässt sich überprüfen, ob ein Suchwert einem Listeneintrag entspricht oder größer als oder kleiner oder gleich einem Listeneintrag ist. Wird zum Beispiel nach einem Eintrag gesucht, bei dem die Variable **value** mit der Relation kleiner oder gleich verknüpft ist, setze den Operator <= direkt vor der ersten Klammer der Liste.

- √ Veränder SimpleLookdown.bs2 durch Einfügen der **value** und **LOOKDOWN** Anweisungen:

```
value = 35
LOOKDOWN value, <= [ 7, 19, 28, 85, 167 ], index
```

- √ Experimentiere mit unterschiedlichen Werten und beobachte, ob die Indexvariable das anzeigt, was du erwartest.

Zusatzfrage: Was passiert, wenn der Wert größer als 167 ist? Diese kleine Wendung im Befehl **LOOKDOWN** kann zu Problemen führen, da der Befehl den Index nicht verändert.

```
' What's a Microcontroller - SimpleLookdown.bs2
' Debug an index using a value and a lookup table.

' {$STAMP BS2}
' {$PBASIC 2.5}

value          VAR     Byte
index          VAR     Nib

value = 167

DEBUG ? value
```

```
LOOKDOWN value, [7, 85, 19, 167, 28], index

DEBUG ? index, CR

DEBUG "Change the value variable to a ", CR,
      "different number in this list:", CR,
      "7, 85, 19, 167, or 28.", CR, CR,

      "Run the modified program and ", CR,
      "check to see what number the ", CR,
      "LOOKDOWN command places in the ", CR,
      "index variable."

END
```

Programmbeispiel: DialDisplay.bs2

Dieses Beispielprogramm überträgt die Position des Schleifers eines Potentiometers durch Aufleuchten der entsprechenden äußeren Segmente auf die 7-Segment Anzeige wie in Abbildung 6-16 gezeigt.

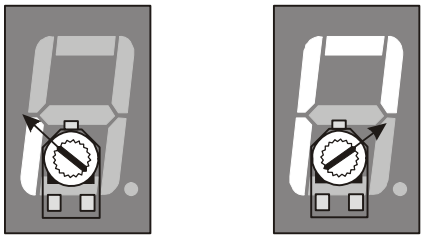


Abbildung 6-16
Anzeige der Schleiferposition eines Potentiometers auf einer 7-Segment LED Anzeige

- √ Führe das folgende Programm aus: DialDisplay.bs2.
- √ Bewege den Schleifer des Potentiometers und überzeuge dich, dass alles funktioniert.
- √ Wenn das Beispielprogramm ausgeführt wird, kann es sein, dass es nicht so genau arbeitet, wie in Abbildung 6-16 dargestellt. Passe die Werte in der Lookdowntabelle so an, dass die Digitalanzeige die Position der Potentiometerstellung präziser wiedergibt.

```
' What's a Microcontroller - DialDisplay.bs2
' Display POT position using 7-segment LED display.

'{$STAMP BS2}
'{$PBASIC 2.5}
```

```
DEBUG "Program Running!"

index          VAR     Nib
time           VAR     Word

OUTH = %00000000
DIRH = %11111111

DO

  HIGH 5
  PAUSE 100
  RCTIME 5, 1, time

  LOOKDOWN time, <= [40, 150, 275, 400, 550, 800], index

  LOOKUP index, [ %11100101, %11100001, %01100001,
                  %00100001, %00000001, %00000000 ], OUTH

LOOP
```


Arbeitsweise von DialDisplay.bs2?

Das Beispielprogramm misst die Entladezeit der RC-Schaltung mit Poti mit dem Befehl **RCTIME** und speichert den Wert in einer Variablen mit Namen **time**.

```
HIGH 5
PAUSE 100
RCTIME 5, 1, time
```

Die **time** Variable wird dann in einer **LOOKDOWN** Tabelle benutzt. Über die **LOOKDOWN** Tabelle entscheidet sich, welcher Eintrag in der Liste kleiner als **time** ist; der entsprechende Positionslistenwert (zwischen 0 und 5 in diesem Fall) wird in der Variablen **index** abgelegt.

```
LOOKDOWN time, <= [40, 150, 275, 400, 550, 800], index
```

Über die Variable **index** wird in der **LOOKUP** Tabelle der entsprechende binäre Wert bestimmt, der dann der Variablen **OUTH** zugewiesen wird.

```
LOOKUP index, [ %11100101, %11100001, %01100001,
                %00100001, %00000001, %00000000 ], OUTH
```

Du bist dran – Ein Segment hinzufügen

DialDisplay.bs2 steuert nur fünf der sechs Segmente an, wenn der Drehknopf bewegt wird. Die Reihenfolge, in der die LEDs über das Programm DialDisplay.bs2 eingeschaltet werden, lautet E, F, A, B, C aber nicht D.

- √ Speicher das Programm DialDisplay.bs2 unter dem neuen Namen DialDisplayYourTurn.bs2 ab.
- √ Veränder DialDisplayYourTurn.bs2 so, dass sich alle sechs äußeren LEDs synchron zur Potentiometereinstellung ein- bzw. ausschalten. Die Segmentreihenfolge sollte wie folgt sein: E, F, A, B, C und D.

ZUSAMMENFASSUNG

In diesem Kapitel haben wir uns mit einer 7-Segment Anzeige befasst und herausgefunden, wie man die Pinbelegung feststellt. Wir haben verschiedene Techniken kennengelernt, die in Verbindung mit sog. Parallel Devices angewendet werden. Mit den Variablen **DIRH** und **OUTH** wurden Variablen eingeführt, über die die BASIC Stamp I/O Pins P8 bis P15 gleichzeitig gesteuert werden können. Über die Befehle **LOOKUP** und **LOOKDOWN** haben wir Möglichkeiten kennengelernt, referenzierend auf Wertelisten zuzugreifen, um Buchstaben- oder Zahlendarstellungen umzusetzen.

Verständnisfragen

1. Aus welchen aktiven Bauteilen besteht eine 7-Segment LED Anzeige?
2. Was versteht man unter einer gemeinsamen Kathode? Was ist damit gemeint, wenn bei einem Display von einer gemeinsame Anode gesprochen wird?
3. Wie bezeichnet man Gruppen von Leiterbahnen, die die Signale von und zu parallelen Bauteilen leiten?
4. Wie lauten die Befehle aus diesem Kapitel, mit denen referenzierend auf Wertelisten zugegriffen wird?

Übungen

1. Wie lautet der **OUTH** Befehl, mit dem die Pins P8, P10, P12 auf HIGH und P9, P11, P13 auf LOW gesetzt werden? Nimm an, dass alle I/O Pins als Eingänge geschaltet sind; gib den **DIRH** Befehl an, der die I/O Pins für die Übertragung von HIGH/LOW Signalen als Ausgänge schaltet, während P14, P15 weiterhin als Eingänge definiert bleiben.
2. Gib die Werte für **OUTH** an, die zur Darstellung der folgenden Buchstaben notwendig sind: a, C, d, F, H, I, n, P, S.

Projekt

1. Schreibe den fortlaufenden Text “FISH CHIPS and dIP” auf der 7-Segment Anzeige. Jeder Buchstabe bleibt 400 ms sichtbar.

Lösungen

Q1. Das aktive Bauteil ist eine LED.
Q2. Gemeinsame Kathode bedeutet, dass die Kathodenanschlüsse aller LEDs in einer 7-Segmentanzeige miteinander verbunden sind, d.h. sie haben einen

gemeinsamen Verbindungspunkt. Entsprechend lässt sich eine gemeinsame Anode erklären.

Q3. Parallelbus.

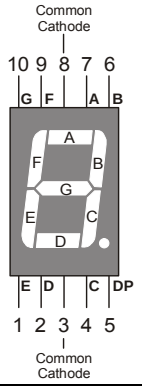
Q4. **LOOKUP** und **LOOKDOWN** bearbeiten Wertelisten.

E1. 1. Schritt: Jede Bitposition, die als **HIGH** deklariert wurde, muss in der Variablen **OUTH** auf „1" gesetzt werden. D.h., dass die Bits 8, 10, und 12 auf „1" und die Bits 9, 11 und 13 entsprechend auf „0" gesetzt werden. **DIRH** wird so konfiguriert, dass die Pins 8, 10, 12, 9, 11, und 13 in der jeweiligen Bitposition mit „1" als Ausgänge definiert werden. 15 und 14 werden Eingänge, deshalb müssen in die Bitpositionen Nullen geschrieben werden. 2. Schritt: Übersetze das Gesagte in PBASIC Anweisungen.

```
Bit   15 14 13 12 11 10  9  8    Bit   15 14 13 12 11 10  9  8
OUTH   0  0  0  1  0  1  0  1    DIRH   0  0  1  1  1  1  1  1

OUTH = %00010101                  DIRH = %00111111
```

E2. Um dieses Problem lösen zu können, müssen wir für jeden Buchstaben die zu aktivierenden Segmente herausfinden. Jedes Segment, das eingeschaltet werden muss, wird auf 1 gesetzt; anschließend wird die entsprechende Binärdarstellung für **OUTH** eingetragen.

Buchst.	LED Segmente	B A F G . C D E	OUTL Wert =
A	e, f, a, b, c, g	1 1 1 1 0 1 0 1	%11110101
C	a, f, e, d	0 1 1 0 0 0 1 1	%01100011
D	b, c, d, e, g	1 0 0 1 0 1 1 1	%10010111
F	a, f, e, g	0 1 1 1 0 0 0 1	%01110001
H	f, e, b, c, g	1 0 1 1 0 1 0 1	%10110101
I	f, e	0 0 1 0 0 0 0 1	%00100001
N	e, g, c	0 0 0 1 0 1 0 1	%00010101
P	alle außer d und c	1 1 1 1 0 0 0 1	%11110001
S	a, f, g, c, d	0 1 1 1 0 1 1 0	%01110110
Aus Abbildung 6-2, p. 160			



P1. Benutze den Schaltplan aus Abbildung 6-10 von Seite 182. Zur Lösung des Problems benutze die Buchstabenmuster aus Übung 1 und 2 und verändere das Programm DisplayDigitsWithLookup.bs2.
In der vorliegenden Lösung wurden die Buchstaben als Konstante deklariert.

```
' What's a Microcontroller - Ch6Prj01_FishAndChips.bs2
' Use a lookup table to store and display digits with a 7-segment display.
' Spell out the message:  FISH CHIPS And dIP

'{$STAMP BS2}
'{$PBASIC 2.5}

' Patterns of 7-Segment Display to create letters
A              CON     %11110101
C              CON     %01100011
d              CON     %10010111
F              CON     %01110001
H              CON     %10110101
I              CON     %00100001
n              CON     %00010101
P              CON     %11110001
S              CON     %01110110
space          CON     %00000000

OUTH = %00000000                                ' All off to start
DIRH = %11111111                                ' All LEDs must be outputs

index          VAR     Byte                     ' 19 chars in message

DO
  DEBUG "index  OUTH   ", CR,
        "-----  --------", CR

  FOR index = 0 TO 18                           ' 19 chars in message

    LOOKUP index, [ F, I, S, H, space, C, H, I, P, S, space,
                    A, n, d, space, d, I, P, space ], OUTH

    DEBUG "   ", DEC2 index, "  ", BIN8 OUTH, CR

    PAUSE 400                                   ' 400 ms between letters

  NEXT
LOOP
```

Weiterführende Bücher

Wie alle Texte, die in diesem Abschnitt erwähnt werden, ist auch dieser aus dem Internet unter der Adresse www.parallax.com kostenlos herunterzuladen

“StampWorks”, Workbook, Version 1.2, Parallax Inc., 2001

StampWorks wurde vom Nuts & Volts Autor Jon Williams geschrieben und behandelt sehr unterschiedliche Experimente. Dieses Buch enthält 32

Experimente, von denen einige auch auf den Gebrauch der 7-Segment Anzeige eingehen. Das Arbeitsbuch stützt sich auf das INEX 1000 Board, das über viele der Komponenten verfügt, die in diesem Abschnitt angesprochen wurden.

Kapitel 7: Lichtstärken messen

LICHTSENSOREN IN GEBRAUCHSGEGENSTÄNDEN

Bisher haben wir zwei verschiedene Arten von Sensoren kennengelernt. Der Taster kann als einfacher Drucksensor interpretiert werden und das Potentiometer als Positions/Dreh-Sensor. In Geräte und Maschinen werden die unterschiedlichsten Sensoren eingebaut, die nicht immer so einfach erkennbar sind, wie Taster und Knöpfe. Temperatur-, Rauch-, Vibrations-, Neigungs- und Lichtsensoren sind weitere Bauteile, die heute in vielen Gebrauchsgegenständen ihren Dienst unsichtbar verrichten.

Ein Beispiel für ein täglich genutztes Gerät mit eingebautem Lichtsensor ist das Fernsehgerät. Wenn es sich über eine Fernbedienung bedienen lässt, verfügt es über einen eingebauten Infrarot-Empfänger, der auf Licht, das mit bloßem Auge nicht wahrgenommen werden kann, reagiert. Über die Fernbedienung wird infrarotes Licht ausgesandt, das Informationen über den gewählten Fernsehkanal, die Lautstärke und viele weitere Funktionen des Fernsehers überträgt. Ein weiteres Beispiel sind Digitalkameras. Mit Hilfe ihres Lichtsensors wird die Belichtung automatisch so gesteuert, dass die Bilder je nach Wetterlage immer gut ausgeleuchtet werden.

DER FOTOWIDERSTAND

Lichtsensoren übernehmen sehr verschiedene Aufgaben; sie haben unterschiedliche Formen, Größen und Preise. Mit speziellen Sensoren lassen sich bestimmte Farbanteile des Lichtes untersuchen, wie zum Beispiel blaues, grünes, rotes oder infrarotes Licht. Einige Sensoren reagieren mehr auf die Helligkeit des Lichtes, während andere auf das emittierte Licht chemischer Reaktionen spezialisiert sind. Die Information, die ein Lichtsensor an den Mikrocontroller sendet, kann unterschiedlich ausfallen. Einige Sensoren geben Spannungswerte aus, andere senden binäre Zahlenfolgen und eine dritte Gruppe reagiert mit unterschiedlichen Widerstandswerten bei unterschiedlichem Lichteinfall.

Der wohl bekannteste, günstigste und am leichtesten zu verstehende Lichtsensor ist der Fotowiderstand (siehe Abbildung 7-1), der bei wechselndem Licht mit unterschiedlichem Widerstandswert reagiert. Er besteht aus Cadmiumsulfid (CdS); ein Stoff, dessen elektrischer Widerstand sich verändert, je nachdem, wieviel Licht er ausgesetzt ist. Helligkeit erzeugt einen geringen, schwaches Licht einen hohen Widerstandswert.

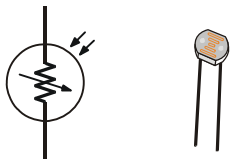


Abbildung 7-1
Fotowiderstand (LDR) Schaltzeichen und Bauteilzeichnung.

Die mit Cadmiumsulfid vergütete lichtempfindliche Oberfläche ist in der Bauteilzeichnung als runde Scheibe zu sehen.

In einer RC-Schaltung kann ein LDR statt eines Potentiometers eingesetzt werden. Das Programm, mit dem der Photowiderstand ausgelesen wird, unterscheidet sich kaum von dem des Potentiometers. Die Übungen dieses Kapitels beschäftigen sich mit Anwendungen, die dem Mikrocontroller über das Licht Informationen zuspielen. Auf diesem Weg werden wir einige PBASIC Programmiertechniken kennenlernen, die sich mit der Datenaufnahme über lange Zeiträume beschäftigen und die die Programme bedienerfreundlicher und lesbarer machen.

ÜBUNG 1: AUFBAU UND TEST EINES LICHTMESSGERÄTES

In dieser Übung wird eine RC-Schaltung mit einem LDR aufgebaut und getestet. Die über einen LDR aufgenommene Entladezeit wird einen Eindruck von der Lichtintensität vermitteln. Wie beim Potentiometertest, werden die über den Befehl **`RCTIME`** gemessenen Zeitwerte auf dem Debug Terminal angezeigt.

Schaltungsbauteile für Lichtmessung

1 Fotowiderstand
1 Widerstand – 220 Ω (rot-rot-braun)
1 Kondensator – 0.01 µF
1 Kondensator – 0.1 µF
1 Drahtbrücke

Obwohl die Bauteileliste zwei Kondensatoren enthält, wird bei jeder Messung nur jeweils ein Kondensator in der Schaltung eingesetzt.

Aufbau einer RC-Schaltung mit Fotowiderstand

Abbildung 7-2 zeigt den Schaltungsaufbau der in diesem Kapitel benutzten RC-Schaltung, Abbildung 7-3 zeigt den zugehörigen Verdrahtungsplan. Die Schaltung unterscheidet sich von der Potentiometerschaltung aus Kapitel 5, Übung 3 in zwei Punkten: 1. Der I/O Pin P2, mit dem die Entladezeit gemessen wird, ist nicht identisch. 2. Der veränderliche Widerstand ist jetzt ein LDR und kein Potentiometer.

√ Baue die Schaltung nach Abbildung 7-2 und Abbildung 7-3 auf.

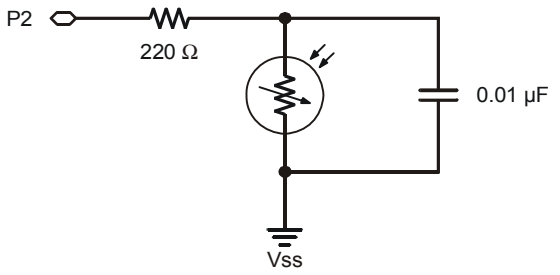


Abbildung 7-2
RC-Schaltung mit Fotowiderstand

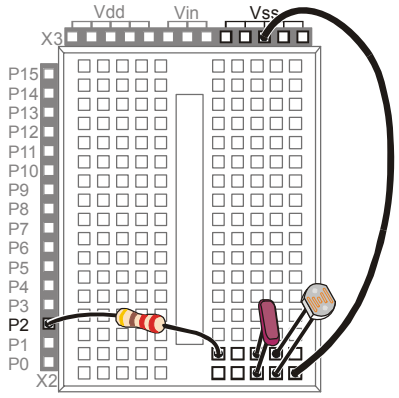


Abbildung 7-3
RC-Schaltung mit Fotowiderstand Verdrahtungsplan

Programmierung der Schaltung mit Fotowiderstand

Das erste Beispielprogramm (TestPhotoresistor.bs2) ist eine leicht veränderte Version von ReadPotWithRcTime.bs2 aus Kapitel 5, Übung 3. Die Potentiometerschaltung aus Kapitel 5 wurde mit I/O Pin P7 verbunden. Die Schaltung dieser Übung wird mit P2 verbunden. Das vorliegende Beispielprogramm muss deshalb angepasst werden. Der Befehl `HIGH 7` (das Potentiometer war mit P7 verbunden), wird jetzt zu `HIGH 2,` da der

LDR mit P2 verbunden ist. Aus dem gleichen Grund muss der Befehl **`RCTIME 7, 1, time`** in **`RCTIME 2, 1, time`** geändert werden.

Programmbeispiel: TestPhotoresistor.bs2

Die Schaltung wird dadurch getestet, dass wir die lichtempfindliche Oberfläche des LDR mit unterschiedlich hellem Licht beleuchten und nicht wie in Kapitel 5, Übung 3 den Schleifer eines Potentiometers bewegen. Läuft das Beispielprogramm fehlerfrei, sollte das Debug Terminal kleine Werte bei starkem Lichteinfall und große Werte bei schwachem Lichteinfall ausgeben.

- √ Gib das Programm TestPhotoresistor.bs2 ein und führe es aus.
- √ Während du den Wert der Variablen **`time`** am Debug Terminal beobachtest, notiere dir den Wert für normale Lichtverhältnisse.
- √ Schalte jetzt die Lichtquellen im Raum aus oder beschatte die Schaltung mit deinen Händen und beobachte wieder die Variable **`time`**. Jetzt sollte der angezeigte Zahlenwert deutlich größer ausfallen. Schreibe den Wert auf.
- √ Wenn die lichtempfindliche Fläche des LDR direkt in die Sonne gehalten wird, sollte die Variable **`time`** einen sehr kleinen Wert anzeigen. Notiere auch diesen Wert.

```
' What's a Microcontroller - TestPhotoresistor.bs2
' Read photoresistor in RC-time circuit using RCTIME command.

' {$STAMP BS2}
' {$PBASIC 2.5}

time           VAR     Word

DO

  HIGH 2
  PAUSE 100
  RCTIME 2, 1, time
  DEBUG HOME, "time =  ", DEC5 time

LOOP
```

Du bist dran – Unterschiedliche Kapazitäten bei unterschiedlichen Lichtverhältnissen

Bei hell ausgeleuchteten Räumen oder Messungen im Freien sollte der 0.01 µF Kondensator gegen einen 0.1 µF Kondensator ausgetauscht werden. Die Zeitmessungen

mit der größeren Kapazität benötigen 10-mal mehr Zeit, das heißt, dass der auf dem Debug Terminal ausgegebene Wert der Variablen **`time`** ebenfalls zehn mal größer ist.

- √ Tausche den 0,01 µF Kondensator gegen einen 0,1 µF Kondensator aus.
- √ Führe das Programm TestPhotoresistor.bs2 erneut aus und überprüfe, ob sich die RC-Zeitmessungen diesmal um den Faktor ~10 geändert haben.
- √ Bevor du mit der nächsten Übung beginnst, tausche die Kondensatoren wieder zurück und baue die Schaltung nach Abbildung 7-2 auf.

ÜBUNG 2: LICHTMESSUNGEN GRAFISCH AUSWERTEN

In der Industrie wird häufig eine große Anzahl an Sensoren bei der Fertigungsüberwachung eingesetzt. Ob es sich um Lichtmessungen in einem Gewächshaus, Flüssigkeitsstände in einer Öl-Raffinerie oder Temperaturmessung in einem Kernreaktor handelt, die Menschen, die all diese Messwerte überwachen müssen und verantwortlich sind für einen fehlerfreien Betrieb, verlassen sich oftmals auf die graphische Darstellung der Sensormesswerte und ziehen daraus die notwendigen Informationen für ihr weiteres Handeln.

Einführung von Stamp Plot Lite

Abbildung 7-4 zeigt die Oberfläche der Software Stamp Plot Lite, auf der mit der BASIC Stamp aufgenommene Entladezeiten dargestellt sind. Der in Abbildung 7-4 dargestellte Graph ist wesentlich leichter zu interpretieren als die 250 Entladezeiten, die diesem Graphen zugrunde liegen. Von links nach rechts wachsen die RC-Zeitmesswerte immer stärker an und fallen dann abrupt ab. Da die Zeitmesswerte größer werden bei geringerem Lichteinfall und kleiner bei stärkerem Lichteinfall, können wir den Graphen erklären: der gemessene Lichteinfall hat kontinuierlich abgenommen um plötzlich wieder stark anzuwachsen.

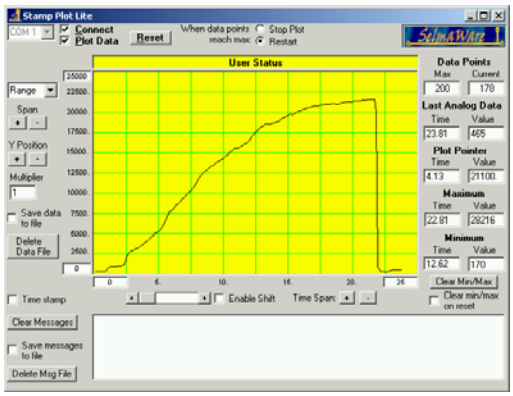


Abbildung 7-4
Stamp Plot Lite
Grafische Darstellung gemessener Lichtstärken

Vielleicht hat ein Mikrocontroller in einem Gewächshaus das künstliche Licht eingeschaltet, nachdem die natürliche Sonneneinstrahlung unter einen Minimalwert gesunken war oder vielleicht hat das Nachführsystem einer Solaranlage sich eingeschaltet und die Paneele zur Sonne neu ausgerichtet, nachdem die gemessene Lichtintensität unter einen bestimmten Wert abgesunken war. Es ließen sich zahlreiche Beispiele für die Interpretation des oben dargestellten Graphen formulieren. Unabhängig davon wirst du aber erkannt haben, wie hilfreich die graphische Darstellung von Daten sein kann.

Stamp Plot Lite ist ein Produkt von SelmaWare Solutions und für Ausbildungszwecke frei erhältlich. Es kann direkt von der Parallax CD auf dem Rechner installiert werden oder von der Parallax Website oder direkt von http://www.selmaware.com/ heruntergeladen werden.

Stamp Plot Lite herunterladen und installieren

Vor dem Herunterladen von Stamp Plot Lite sollte WinZip auf dem PC oder Laptop installiert sein. WinZip kann auch von der Parallax CD installiert oder von www.winzip.com heruntergeladen werden. Es folgen die Anweisungen zum Herunterladen und Installieren von Stamp Plot Lite von der Parallax Website:

- √ Rufe unter www.parallax.com den Downloadbereich auf.
- √ Wähle aus: BASIC Stamp Software.
- √ Lade folgende Datei herunter: "**Stamp Plot Lite** graphing utility…"
- √ Speicher die Datei StampPlot.zip auf der Festplatte.
- √ Aktiviere mit einem Doppelklick den Entpackungsvorgang von StampPlot.zip.

- √ Wird der WinZip wizard benutzt, muss man nur den Hinweisen folgen und das Installationsprogramm starten. In der WinZip classic Einstellung wird das Installationsprogramm mit einem Doppelklick auf Setup.exe gestartet.
- √ Folge den Installationsanweisungen von Stamp Plot Lite.

Programmierung: Messwerte an Stamp Plot Lite übertragen

Messwerte an Stamp Plot Lite zur graphischen Auswertung zu übertragen ist vergleichbar mit der Aufgabe, Messwerte auf dem Debug Terminal auszugeben; es müssen nur einige Regeln beachtet werden: Messwerte dürfen nur den Formatierungsbefehl **`DEC`** und das Kontrollzeichen **`CR`** enthalten. Wenn wir die Werte der Variablen **`time`** darstellen wollen, müssen wir sie auf dem Debug Terminal als Dezimalzahl gefolgt von einem Zeilenvorschub (CR) ausgeben.

```
DEBUG DEC time, CR
```

Darstellungshinweise werden in Anführungszeichen gesetzt und lassen sich so ebenfalls an Stamp Plot Lite übertragen. Wir bezeichnen sie als Kontrollzeichen. Kontrollzeichen werden an Stamp Plot Lite am Anfang eines PBASIC Programmes übergeben. Einstellungen lassen sich aber auch direkt über die Software vornehmen; häufig ist es aber einfacher diese Aufgabe über die BASIC Stamp zu erledigen. Es folgt ein Beispiel zur Konfigurationseinstellung aus dem folgenden Beispielprogramm, mit dem die RC-Zeitmessungen leichter zu lesen sind, ohne das die Darstellungseigenschaften von Stamp Plot Lite verändert wurden.

```
DEBUG "!AMAX 1250", CR,
      "!TMAX 25", CR,
      "!TMIN 0", CR,
      "!SHFT ON", CR,
      "!RSET",CR
```

Weitere Informationen darüber, wie Werte und Kontrollzeichen an Stamp Plot Lite übertragen werden, findet man in der Stamp Pot Lite Help Datei. Klicke auf Start, wähle Alle Programme - Stamp Plot - Stamp Plot Help.

Programmbeispiel: PlotPhotoresistor.bs2

Um die Daten des Fotowiderstandes graphisch darzustellen, folge den Anweisungen:

- √ Führe das Programm PlotPhotoresistor.bs2 aus.
- √ Überzeuge dich davon, dass auf dem Debug Terminal eine Spalte mit Zahlenwerten erscheint. Abbildung 7-5 zeigt ein Beispiel.

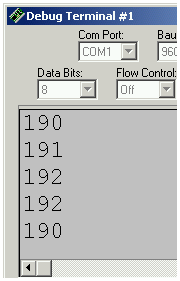


Abbildung 7-5
Beispiel von Zahlenwerten auf dem Debug Terminal.

√ Notiere dir den COM Port Eintrag im COM Port Feld oben links im Debug Terminal.

√ Starte über das Windows Startmenü Stamp Plot Lite. Klicke auf *Start*, wähle dann *Programme → Stamp Plot → Stamp Plot Lite*.

√ Wähle im COM Port Feld den gleichen COM Port wie im Debug Terminal. Abbildung 7-6 zeigt ein Beispiel mit dem Eintrag COM1 im Debug Terminal, deshalb wurde unter Stamp Plot auch COM1 eingestellt. Auf deinem Rechner kann ein anderer Bezeichner erscheinen, der dann auch unter Stamp Plot eingetragen werden muss.

√ Schließe das Debug Terminal (klicke auf *X* oben rechts im Fenster oder klicke auf *Close* im unteren Fenster).

√ Markiere im Fenster von Stamp Plot Lite *Connect* und *Plot Data*.

Abbildung 7-6
COM Port Einstellungen

Debug Terminal (links) und Stamp Plot Lite (rechts).

√ Drücke die Resettaste auf dem Board of Education oder HomeWork Board und starte damit das BASIC Stamp Programm erneut. Die `DEBUG` Befehle werden an das Programm Stamp Plot Lite übergeben.

√ Sobald PlotData angeklickt wird, werden die Daten graphisch dargestellt. Halte eine Hand in verschiedenen Abständen über den Fotowiderstand und simuliere damit unterschiedliche Beleuchtungsstärken. Denke daran, je stärker

die Abdunklung des Fotowiderstandes, umso höher der aufgezeichnete Funktionswert, je geringer die Abdunklung, umso kleiner der Funktionswert.

HINWEIS: Ein COM Port kann nur von einem Programm zur Zeit genutzt werden.

Bevor versucht wird, über den BASIC Stamp Editor ein weiteres Programm auszuführen, müssen die Markierungsfelder *Connect* und *Plot Data* in Stamp Plot Lite deaktiviert werden.

Wird die Verbindung mit Stamp Plot Lite wiederhergestellt (durch Anklicken der *Connect* und *Plot Data* Markierungsfelder), muss das Debug Terminal geschlossen werden.

```
' What's a Microcontroller - PlotPhotoresistor.bs2
' Graph light levels using Stamp Plot Lite.

' {$STAMP BS2}
' {$PBASIC 2.5}

time           VAR     Word

DEBUG "!AMAX 1250", CR,
      "!TMAX 25", CR,
      "!TMIN 0", CR,
      "!SHFT ON", CR,
      "!RSET",CR
DO

  HIGH 2
  PAUSE 100
  RCTIME 2, 1, time
  DEBUG DEC time, CR

LOOP
```


Du bist dran – Die Darstellung anpassen

Der Maßstab auf der horizontalen (time span) und vertiaklen Achse (span) lässt sich über die zugeordneten + und – Tasten anpassen. Für die Ordinate befinden sich die Tasten links oben neben der Achsbeschriftung. Der Time Span wird über die Tasten unterhalb der Abszisse eingestellt.

√ Experimentiere mit den Einstellungen, indem die Werte erhöht und erniedrigt werden und beobachte gleichzeitig die sich einstellenden Effekte bei der graphischen Darstellung.

Treten Schwierigkeiten bei der graphischen Darstellung auf, lassen sich sowohl das Board of Education als auch das BASIC Stamp HomeWork Board über den Reset-Schalter neu starten und die Anfangseinstellungen zurückholen.

ÜBUNG 3: LICHTEREIGNISSE AUFZEICHNEN

Der Programmspeicher einer BASIC Stamp verliert selbst dann, wenn die Spannungsversorgung abgeschaltet wird, seine Daten nicht. Das ist eine besonders nützliche Eigenschaft. Sobald die Spannungsversorgung wieder angelegt wird, startet das Programm erneut. Die Teile eines Programmspeichers, die vom Programm nicht genutzt werden, lassen sich als Datenspeicher einsetzen. Dies gilt vornehmlich für Daten, die die BASIC Stamp ständig benötigt und die auch nach einem Spannungsausfall sofort wieder verfügbar sein müssen.

In Abbildung 7-7 ist der Chip markiert, in dem Programme und Daten abgelegt werden. Dieser Chip wird als EEPROM bezeichnet. EEPROM ist ein Akronym und steht für *electrically erasable programmable read-only memory*. Umgangssprachlich reden wir kurz von "E-E-Prom".

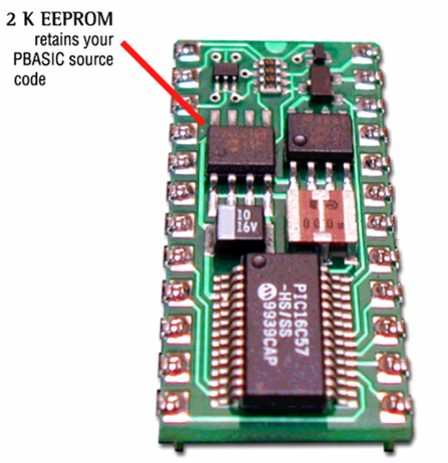


Abbildung 7-7
EEPROM Chip einer BASIC Stamp

Abbildung 7-8 zeigt den Inhalt eines EEPROMs in Form einer sogenannten Speicherbelegung. Das Fenster wird über *Run* und *Memory Map* aufgerufen. Die Speicherbelegung verfügt über unterschiedliche Farbdarstellungen, die dem Nutzer sofort zeigen, wie der RAM- (Variable) und der EEPROM-Bereich (Programmspeicher) der

BASIC Stamp genutzt werden. Die *EEPROM Map* zeigt zwei Graphen. Der Balken ganz links im Fenster zeigt mit seiner Höhe an, wie viel vom Programmspeicher für das Fotowiderstandsprogramm aus Übung 2: Lichtmessungen grafisch auswerten belegt ist. Die *EEPROM Map* rechts daneben zeigt mit den blau markierten Bytes an, dass 101 von insgesamt 2048 Bytes des EEPROMS vom Programm belegt sind. Die restlichen 1947 Bytes stehen als Datenspeicher zur Verfügung.

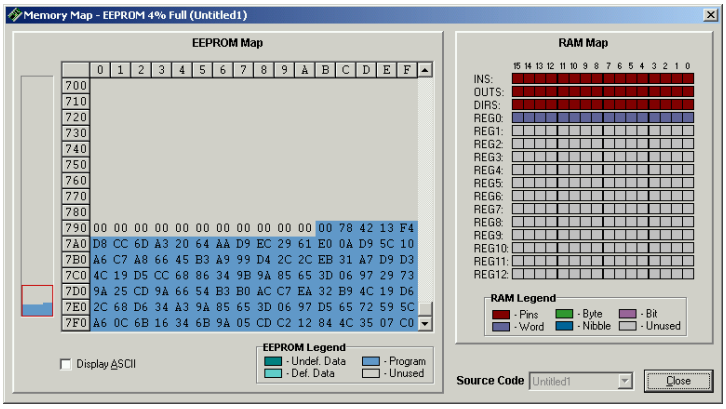


Abbildung 7-8
Speicherbelegung

Das Fenster erscheint durch Anklicken von Run und Memory Map.

7

2048 Byte = 2 KB. In 2048 Byte lassen sich 2048 unterschiedliche Zahlen speichern, von denen jede einen Wert zwischen 0 und 255 haben kann.

Der Buchstabe „B“ steht abkürzend für Byte. Das kleine „b“ steht für Bit. Beachte den Unterschied: 2048 Kb bedeutet, dass 2048 unterschiedliche Zahlen, jede vom Wert 0 oder 1 gespeichert werden können.

Sowohl „K” als auch „k” stehen abkürzend für „Kilo”. Und trotzdem mag es einen kleinen Unterschied geben. In der Datenverarbeitung wird für das binäre Kilobyte (dass sind $1 \times 2^{10} = 1024$) gerne der Großbuchstabe “K” verwendet. Bezieht man sich auf genau 2000 Byte, dann sollte das kleine „k“ verwendet werden, es steht für Kilo ($1 \text{ X } 10^3 = 1000$) des metrischen Systems.

Die Datenspeicherung im EEPROM kann zum Beispiel bei Datenaufzeichnungen über einen längeren Zeitraum sehr nützlich sein. Anwendungsbeispiele sind: Temperaturaufzeichnungen in einem LKW mit tiefgekühlten Lebensmitteln oder in einer Wetterstation. Aber auch Lichtstärkemessungen ließen sich mit einer Wetterstation aufzeichnen.

Da wir einen Fotowiderstand zur Lichtmessung benutzen, werden wir in dieser Übung eine Technik kennenlernen, die Messdaten in einem EEPROM abspeichert und von dort auch wieder zurücklädt. Wir werden dazu zwei PBASIC Beispielprogramme entwerfen, von denen das erste die aufgenommenen Messdaten im EEPROM abspeichert und das zweite diese Daten wieder ausliest und auf dem Debug Terminal anzeigt.

Programmierung: Datenaufzeichnung von Langzeitmessungen

Mit dem Befehl **`WRITE`** werden Daten in das EEPROM geschrieben und mit **`READ`** wieder ausgelesen.

Die Syntax des Befehls **`WRITE`** lautet:

WRITE *Location, {WORD} Data Item*

Soll zum Beispiel die Zahl 195 in die Adresse 7 des EEPROMS geschrieben werden, dann erfolgt dies über den Befehl:

```
WRITE 7, 195
```

Variablen vom Typ Word können Zahlenwerte zwischen 0 und 65565 verwalten, während die vom Typ Byte auf den Bereich zwischen 0 und 255 beschränkt sind. Eine Word Variable benötigt im Speicher zwei Byte. Soll eine Variable vom Typ Word in einem EEPROM abgelegt werden, muss der Bezeichner **`Word`** der Zahl vorangestellt werden. Da eine Word-Variable zwei Byte benötigt, setzt sich der Schreibvorgang an der übernächsten Speicherstelle fort. Sollen beispielsweise die Werte 659 und 50012 im EEPROM abgelegt werden, dann könnte zum Beispiel die erste Zahl an der Speicherstelle 8 abgelegt werden. Der zweite Wert muss dann aber an der Adresse 10 beginnend gespeichert werden.

```
WRITE 8, Word 659
WRITE 10, Word 50012
```

Ist es möglich ein Programm zu überschreiben? Ja, und wenn es passiert, reagiert das Programm fehlerhaft oder kann sich "aufhängen". Da die PBASIC Programmtoken an den obersten EEPROM-Adressen abgelegt werden, sollten Zahlen über den **`WRITE`** Befehl immer in den untersten Adressbereichen des EEPROMS geschrieben werden.

Wie finde ich heraus, ob die benutze *`Adresse`* zu groß ist? Über die Memory map kannst du den höchsten Adresswert herausfinden, der noch nicht vom PBASIC Programm benutzt wird. Die Zahlen an der linken Seite der Memory Map (siehe Abbildung 7-8 auf Seite 211) geben in hexadezimaler Schreibweise an, welche Speicher noch nicht vom Programm benutzt werden. 79A ist dort beispielsweise der höchste Wert für einen freien Speicher. Hexadezimal heißt 16-er System, so dass 79A bedeutet:

$$7\times16^2 + 9\times16^1 + A\times16^0$$

$$= 7\times16^2 + 9\times16^1 + 10\times16^0$$

$$= 7\times256 + 9\times16 + 10\times1$$

$$= 1946.$$

Warum ergibt sich 1946 und nicht 1947? Insgesamt gibt es 1947 freie Speicher, die von 0 bis zur Adresse 1946 adressiert werden. Mit Hilfe des **`DEBUG`** Befehls und der Formatierungsanweisung **`DEC`** und dem Hexadezimaloperator **`$`** ist die BASIC Stamp in der Lage, die Umrechnungen selbst vorzunehmen:

```
DEBUG DEC $79A
```

Hilfe zum Hexadezimalsystem findest du im Informationsfenster auf Seite 189.

Beispielprogramm: StoreLightMeasurementsInEeprom.bs2

In diesem Beispielprogramm erfährst du, wie man den **`WRITE`** Befehl einsetzt. Dazu werden alle 5 Sekunden über eine Dauer von 2,5 Minuten Lichtmessungen vorgenommen und die Werte ins EEPROM geschrieben.

- √ Gib das folgende Programm ein und führe es anschließend aus: StoreLightMeasurementsToEeprom.bs2 .
- √ Zeichne die auf dem Debug Terminal ausgegebenen Messwerte auf, so dass die aus dem EEPROM zurückgelesenen Werte überprüft werden können.

Dunkle den Fotowiderstand während der Messung langsam immer stärker ab. Die Messung sollte insgesamt 2,5 Minuten dauern, um ausreichend Datenmaterial zu haben.

```
' What's a Microcontroller - StoreLightMeasurementsInEeprom.bs2
' Write light measurements to EEPROM.
```

```
' {$STAMP BS2}
' {$PBASIC 2.5}

time           VAR     Word
eepromAddress  VAR     Byte

DEBUG "Starting measurements...", CR, CR,
      "Measurement   Value", CR,
      "-----------   -----", CR

PAUSE 1000

FOR eepromAddress = 0 TO 58 STEP 2
  HIGH 2
  PAUSE 5000
  RCTIME 2, 1, time
  DEBUG DEC2 eepromAddress,
        "            ", DEC time, CR
  WRITE eepromAddress, Word time
NEXT

DEBUG "All done.  Now, run:", CR,
      "ReadLightMeasurementsFromEeprom.bs2"
END
```

Arbeitsweise von StoreLightMeasurementsInEeprom.bs2

Die **FOR…NEXT** Schleife, die die Entladezeiten aufnimmt und im EEPROM ablegt, zählt in Zweierschritten, da die Werte im EEPROM im Format Word abgelegt werden.

```
FOR eepromAddress = 0 to 58 STEP 2
```

Der Befehl **RCTIME** lädt die Zeitmesswerte in die Variable **time** vom Typ Word.

```
RCTIME 2, 1, time
```

Die **time** Variable wird bei jedem Schleifendurchlauf an der Adresse, die durch den aktuellen Wert der Variablen **eepromAddress** bestimmt wird, abgelegt. **eepromAddress** erhöht sich bei jedem Schleifendurchlauf um zwei, da die im Speicher abgelegten Zahlen vom Typ Word (2 Byte) sind. Die Adresse in einem **WRITE** Befehl wird immer in Byte angegeben.

```
WRITE eepromAddress, Word time
NEXT
```

Programmierung: Daten auslesen

Mit dem Befehl **READ** werden Daten aus dem EEPROM ausgelesen. Die Syntax des **READ** Befehls lautet:

READ ***Location, {WORD} Data Item***

Während der **WRITE** Befehl sowohl Konstante als auch Variable verarbeitet, muss das **DataItem** Argument des **READ** Befehls immer eine Variable sein, da es Werte aus dem EEPROM ausliest, die über den **READ** Befehl dort hineingeschrieben worden sind.

Nehmen wir an, dass die Variablen **eepromValueA** und **eepromValueB** Variablen vom Typ **Word** sind und **littleEE** eine Variable vom Typ Byte ist. Mit den folgenden Befehlen lassen sich die abgespeicherten Werte wieder zurückholen

```
READ 7, littleEE
READ 8, Word eepromValueA
READ 10, Word eepromValueB
```


Programmbeispiel: ReadLightMeasurementsFromEeprom.bs2

Im Beispielprogramm wird gezeigt, wie der Befehl **READ** aufgerufen werden muss, um Daten der Lichtmessung, die mit dem Programm StoreLightMeasurementsInEeprom.bs2 im EEPROM abgespeichert wurden, wieder auszulesen.

- √ Sobald das Programm StoreLightMeasurementsToEeprom.bs2 beendet ist, trenne die BASIC Stamp von der Spannungsversorgung um sie gleich wieder anzuschließen und zu prüfen, ob die Daten im EEPROM nicht gelöscht wurden. Verfahre mit dem BASIC Stamp Editor ebenso, beende das Programm und starte es anschließend erneut.
- √ Die Spannungsversorgung der BASIC Stamp bleibt solange abgeschaltet, bis das Programm ReadLightMeasurementsFromEeprom.bs2 einsatzbereit ist.
- √ Führe das Programm ReadLightMeasurementsFromEeprom.bs2 aus.

Vergleiche die angezeigte Tabelle dieses Programmes mit den Werten des Programms StoreLightMeasurementsInEeprom.bs2; die Werte sind die gleichen.

```
' What's a Microcontroller - ReadLightMeasurementsFromEeprom.bs2
' Read light measurements from EEPROM.

' {$STAMP BS2}
' {$PBASIC 2.5}
```

```
time           VAR     Word
eepromAddress  VAR     Byte

DEBUG "Retrieving measurements", CR, CR,
      "Measurement   Value", CR,
      "----------   -----", CR

FOR eepromAddress = 0 TO 58 STEP 2
  READ eepromAddress, Word time
  DEBUG DEC2 eepromAddress, "           ", DEC time, CR
NEXT

END
```

Arbeitsweise von ReadLightMeasurementsFromEeprom.bs2

Die Befehle **WRITE** und **READ** benutzen Byteadressen. Da aus dem EEPROM Werte vom Typ Word ausgelesen werden, muss die Variable **eepromAddress** nach jedem **FOR…NEXT** Schleifendurchlauf um 2 erhöht werden.

```
FOR eepromAddress = 0 to 58 STEP 2
```

Mit dem **READ** Befehl wird der Wert vom Typ Word an der Adresse **eepromAddress** ausgelesen. Sein Wert wird der Variablen **time** zugewiesen.

```
READ eepromAddress, Word time
```

Die Werte der Variablen **time** und **eepromAddress** werden spaltenweise in einer Tabelle auf dem Debug Terminal ausgegeben.

```
   DEBUG DEC2 eepromAddress, "           ", DEC time, CR
NEXT
```

Du bist dran – Gespeicherte Daten graphisch darstellen

√ Änder das Programm ReadLightMeasurementsFromEeprom.bs2 so ab, dass es die Daten über Stamp Plot Lite ausgibt. Achte darauf, dass die **DEBUG** Anweisung nur den Wert gefolgt von einem Zeilenrücklauf (CR) enthält.

ÜBUNG 4: EINFACHES LICHTMESSGERÄT

Die Information eines Lichtsensors kann auf verschiedene Art und Weise ausgewertet und interpretiert werden. Das Lichtmessgerät aus dieser Übung ändert seine Umlauffrequenz in Abhängigkeit von der jeweils gemessenen Lichtintensität.

Bauteile des Lichtmessgerätes

1 Fotowiderstand
1 Widerstand – 220 Ω (rot-rot-braun)
1 Kondensator – 0.01 µF
1 Kondensator – 0.1 µF
1 7-Segment LED Anzeige
8 Widerstände – 1 kΩ (braun-schwarz-rot)
6 Verbindungsdrähte

Schaltungsaufbau Lichtmessgerät

Abbildung 7-9 zeigt den Schaltungsaufbau einer 7-Segment Anzeige und einer Schaltung mit Fotowiderstand, die im Lichtmessgerät benötigt werden; Abbildung 7-10 zeigt den zugehörigen Verdrahtungsplan. Die Schaltung mit Fotowiderstand haben wir bereits in den letzten beiden Übungen kennengelernt; die Schaltung mit der 7-Segment Anzeige wurde in Kapitel 6 vorgestellt und von der BASIC Stamp gesteuert.

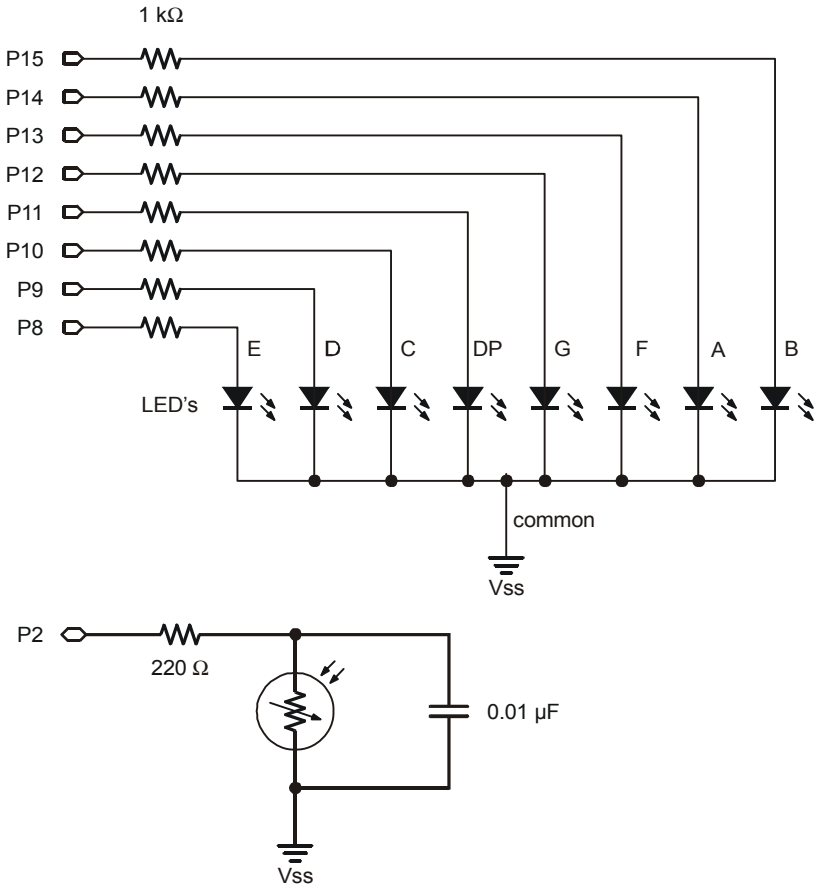


Abbildung 7-9
Schaltungsaufbau für Lichtmess-gerät

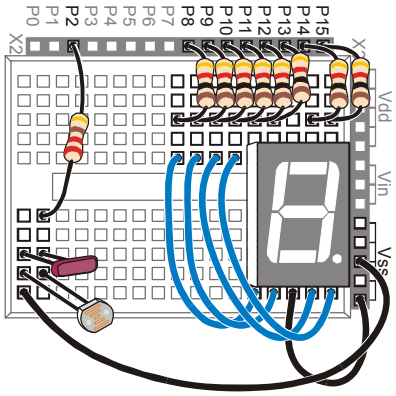


Abbildung 7-10
Verdrahtungsplan
für Abbildung 7-9

√ Baue die Schaltung auf, wie in Abbildung 7-9 und Abbildung 7-10 gezeigt.
√ Überprüfe die 7-Segment LED Anzeige auf Funktionstüchtigkeit mit Hilfe des Programms SegmentTestWithHighLow.bs2 aus Kapitel 6, Übung 2 und stelle so sicher, dass alles korrekt verdrahtet ist.

Unterprogramme

Die meisten Programme, die wir bisher geschrieben haben, sind in eine `DO…LOOP` Schleife eingebettet. In ihr wurden die grundlegenden Aktionen oder Routinen ausgeführt, deshalb nennen wir diesen Programmteil auch Hauptprogramm. Sobald zu einer elektrischen Schaltung weitere Schaltkreise hinzugefügt werden und damit dem Hauptprogramm weitere Programmfunktionen hinzugefügt werden, die die neuen Schaltkreise abfragen, kann ein sehr unübersichtliches umfangreiches Hauptprogramm entstehen. Ein Programm ist übersichtlicher gestaltet, wenn es in kleinere Teilprogramme, von denen jedes nur eine Teilaufgabe übernimmt, zerlegt wird. PBASIC stellt dafür einige Befehle zur Verfügung. Mit einem Sprungbefehl gelangt man aus dem Hauptprogramm in ein Unterprogramm und die dort anstehenden Aufgaben werden abgearbeitet. Anschließend kehrt das Programm zum Hauptprogramm zurück und setzt seine Arbeit fort. Teilaufgaben lassen sich über eine solche Struktur sehr einfach aus dem Hauptprogramm herausnehmen und es damit übersichtlicher und lesbarer werden.

Abbildung 7-11 zeigt ein Beispiel für ein Unterprogramm und wie es aufgerufen wird. Mit `GOSUB Subroutine_Name` wird das Programm angewiesen, mit seiner Ausführung im Unterprogramm mit der Einsprungmarke `Subroutine_Name:` fortzufahren. Es werden alle folgenden Befehle und Anweisungen ausgeführt, bis das Programm auf die

Anweisung **RETURN** trifft. An dieser Stelle endet das Unterprogramm und die Arbeit wird im Hauptprogramm in der Zeile fortgesetzt, die auf den **GOSUB** Befehl folgt. Für das Beispiel in Abbildung 7-11 heißt das, der nächste auszuführende Befehl nach dem Rücksprung aus dem Unterprogramm lautet: **DEBUG "Next command"**.

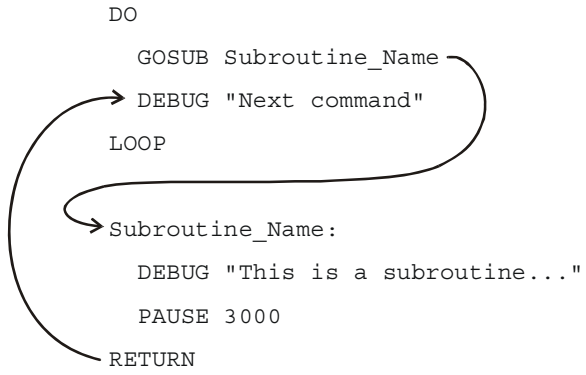


Abbildung 7-11
Wie ein Unterprogrammaufruf abgearbeitet wird

Was ist eine Einsprungmarke? Eine Einsprungmarke – kurz auch Marke - ist ein Name, der wie ein Platzhalter in einem Programm eingesetzt werden kann. **GOSUB, GOTO**, **ON GOTO** und **ON GOSUB** sind Befehle, mit denen zu einer Einsprungmarke im Programm gesprungen werden kann. Eine Marke wird immer mit einem Doppelpunkt abgeschlossen; setzt sich eine Marke aus mehreren Wörtern zusammen, werden die einzelnen Worte mit einem Unterstrich verbunden. Reservierte Worte dürfen nicht als Marke verwendet werden. Alle anderen Regeln, die es bei Bezeichnung von Marken zu beachten gilt, entsprechen denen bei Variablennamen (siehe Informationskasten auf Seite 55).

Programmbeispiel: SimpleSubroutines.bs2

In diesem Beispielprogramm wird gezeigt, wie aus Unterprogrammen Mitteilungen an das Debug Terminal gesandt werden.

- √ Schaue dir das Programm SimpleSubroutines.bs2 an und finde heraus, in welcher Reihenfolge die **DEBUG** Befehle abgearbeitet werden.
- √ Führe das Programm aus.
- √ Vergleiche das Verhalten des Programms mit den eigenen Vorhersagen.

```
' What's a Microcontroller - SimpleSubroutines.bs2
```

```
' Demonstrate how subroutines work.

' {$STAMP BS2}
' {$PBASIC 2.5}

DO

  DEBUG CLS, "Start main routine.", CR
  PAUSE 2000
  GOSUB First_Subroutine
  DEBUG "Back in main.", CR
  PAUSE 2000
  GOSUB Second_Subroutine
  DEBUG "Repeat main...", CR
  PAUSE 2000

LOOP


First_Subroutine:

  DEBUG "  Executing first "
  DEBUG "subroutine.", CR
  PAUSE 3000

RETURN


Second_Subroutine:

  DEBUG "  Executing second "
  DEBUG "subroutine.", CR
  PAUSE 3000

RETURN
```


Arbeitsweise von SimpleSubroutines.bs2

Abbildung 7-12 zeigt, wie der erste Unterprogrammaufruf **`First_Subroutine`** im Hauptprogram (**`DO…LOOP`**) arbeitet. Die Anweisung **`GOSUB First_Subroutine`** verursacht einen Sprung zur Marke **`First_Subroutine:`**. Die drei Befehle des Unterprogramms werden ausgeführt. Mit Erreichen der Anweisung **`RETURN`** kehrt das Programm zu der Anweisung zurück, die dem Befehl **`GOSUB`** folgt, in diesem Fall ist das **`DEBUG "Back in Main.", CR`**.

Was ist ein Unterprogrammaufruf? Wird über den Aufruf des Befehls **`GOSUB`** zu einem Unterprogramm gesprungen, sprechen wir von einem Unterprogrammaufruf.

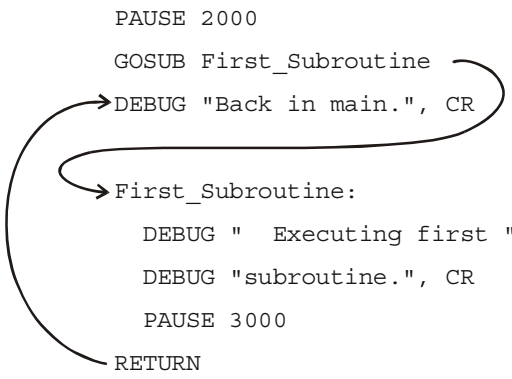


Abbildung 7-12
Erster Unterprogrammaufruf

Abbildung 7-13 zeigt ein zweites Beispiel des gleichen Vorgangs mit einem zweiten Unterprogrammaufruf (**`GOSUB Second_Subroutine`**).

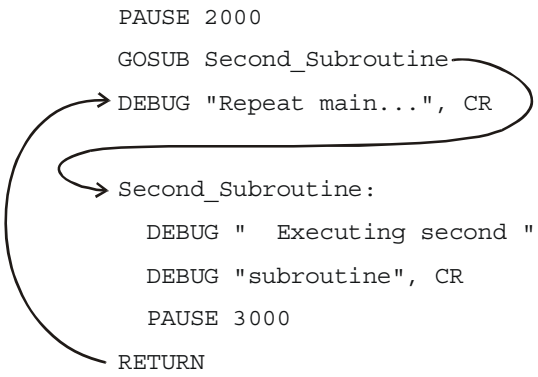


Abbildung 7-13
Zweiter Unterprogrammaufruf

Du bist dran – Unterprogramme hinzufügen und verschachteln

Es lassen sich weitere Unterprogramme hinzufügen und aus dem Hauptprogramm aufrufen.

- √ Füge das Unterprogrammbeispiel aus Abbildung 7-11 auf Seite 220 dem Programm SimpleSubroutines.bs2 hinzu.
- √ Änder die **DEBUG** Befehle entsprechend ab, so dass alle drei Unterprogramme ihre Informationen entsprechend ausgeben können.

Es lassen sich auch Unterprogramme aus Unterprogrammen aufrufen. Dies bezeichnen wir als geschachtelte Unterprogramme.

- √ Verschiebe die **GOSUB** Anweisung mit dem Unterprogrammaufruf **Subroutine_Name** in eine der anderen Unterprogramme und beobachte, wie sie arbeitet.

Bei geschachtelten Unterprogrammen sind maximal vier Schachtelungsebenen zugelassen. Weitere Informationen gibt es unter den Stichworten **GOSUB** und **RETURN** im BASIC Stamp Manual.

Unterprogrammtechnik für Lichtmessgerät

Die einzelnen Anzeigensegmente werden kreisförmig durchlaufen, wobei die Umlaufgeschwindigkeit um so höher ist, je höher die vom LDR aufgenommene Lichtintensität ist. Wird das Licht gedimmt, nimmt auch die Umlauffrequenz der 7-Segment Anzeige ab.

Das Programm für das Lichtmessgerät enthält drei unterschiedliche Ausführungsteile:

1. Fotowiderstand auslesen.
2. Berechnung, wie groß die Pause sein muss, bis die 7-Segment LED Anzeige aktualisiert wird.
3. Aktualisierung der 7-Segment LED Anzeige.

Für jedes dieser Ausführungsteile wurde ein eigenes Unterprogramm geschrieben. Die Hauptroutine **DO…LOOP**, die hier als Endlosschleife installiert ist, ruft diese Unterprogramme nacheinander auf.

Programmbeispiel: LightMeter.bs2

Eindeutige Beleuchtungsbedingungen machen den Unterschied aus. Um beste Ergebnisse zu erzielen, führe diesen Test in einem von Neonröhren beleuchteten Raum durch, ohne direkte Sonneneinstrahlung (verdunkle die Jalousien). Informationen darüber, wie das Messgerät unter anderen Lichtbedingungen einzustellen ist, findest du unter der Rubrik „Du bist dran ...".

√ Führe das Programm LightMeter.bs2 aus.

√ Überzeuge dich davon, dass das auf der 7-Segment Anzeige dargestellte Beleuchtungsmuster durch die vorherrschenden Beleuchtungsbedingungen, die über den LDR-Sensor aufgenommen werden, gesteuert wird, indem du den Sensor mit der Hand oder einem Stück Papier unterschiedlich intensiv beschattest.

```
' What's a Microcontroller - LightMeter.bs2
' Indicate light level using 7-segment display.

' {$STAMP BS2}
' {$PBASIC 2.5}

DEBUG "Program Running!"

index          VAR     Nib                  ' Variable declarations.
time           VAR     Word

OUTH = %00000000                            ' Initialize 7-segment display.
DIRH = %11111111

DO                                          ' Main routine.

  GOSUB Get_Rc_Time
  GOSUB Delay
  GOSUB Update_Display

LOOP
                                            ' Subroutines
Get_Rc_Time:                                ' RC-time subroutine

  HIGH 2
  PAUSE 3
  RCTIME 2, 1, time

RETURN

Delay:                                      ' Delay subroutine.
```

```
  PAUSE time

RETURN

Update_Display:                              ' Display updating subroutine.

  IF index = 6 THEN index = 0
  '                BAFG.CDE
  LOOKUP index, [ %01000000,
                  %10000000,
                  %00000100,
                  %00000010,
                  %00000001,
                  %00100000 ], OUTH
  index = index + 1

RETURN
```


Arbeitsweise von LightMeter.bs2

In den ersten beiden Zeilen werden die Variablen deklariert. Dabei spielt es keine Rolle, ob diese Größen im Haupt- oder Unterprogramm aufgerufen werden. Wir vereinbaren, dass die Variablen und Konstanten am Anfang eines Programms, dem sogenannten Deklarationsteil, definiert werden. Dieser Begriff erscheint als Kommentar rechts in der ersten Zeile der Variablendeklaration.

```
index VAR Nib                    ' Deklarationsteil.
time  VAR Word
```

In vielen Programmen werden auch Deklarationen benötigt, die man einmal am Anfang eines Programms einstellen sollte. Zum Beispiel, wenn bei einer 7-Segment Anzeige alle I/O Pins auf LOW gesetzt und als Ausgänge deklariert werden müssen. Diesen Abschnitt innerhalb eines PBASIC Programms bezeichnen wir als Initialisierung.

```
OUTH = %00000000                 ' Initialisierung der 7-Segment
DIRH = %11111111                 ' Anzeige
```

Der folgende Programmteil wird als Hauptprogramm bezeichnet. Im Hauptprogramm wird als erstes Unterprogramm **`Get_Rc_Time`**, dann das Unterprogramm **`Delay`** und schließlich **`Update_Display`** aufgerufen. Denke bitte daran, dass das Programm die drei Unterprogramme beliebig oft durchläuft und abarbeitet.

```
DO                               ' Hauptprogramm.
  GOSUB Get_Rc_Time
```

```
  GOSUB Delay
  GOSUB Update_Display
LOOP
```

Alle Unterprogramme werden im allgemeinen über das Hauptprogramm aufgerufen. Das erste Unterprogramm trägt den Namen **Get_Rc_Time:** und misst im Fotowiderstand-Schaltkreis die Entladezeit. In diesem Unterprogramm wird der Befehl **PAUSE** aufgerufen, der dafür sorgt, dass sich der Kondensator aufladen kann. Die ***Duration*** Wert dieses Befehls ist klein, da die Pause nur so lang sein muss, das sich der Kondensator aufladen kann. Beachte, dass der Befehl **RCTIME** den Wert der Variablen **time** bestimmt. Dieser Wert wird im zweiten Unterprogramm benötigt.

```
                                ' Unterprogramme
Get_Rc_Time:                    ' Entladezeit
  HIGH 2
  PAUSE 3
  RCTIME 2, 1, time
RETURN
```

Der Name der zweiten Routine lautet **Delay**, das Unterprogramm selbst besteht nur aus dem Befehl **PAUSE**. Sind zusätzliche Berechnungen mit der Variablen **time** erforderlich, bevor sie im Befehl **PAUSE** aufgerufen werden, sollte dies in diesem Unterprogramm erfolgen.

```
Delay:
  PAUSE time
RETURN
```

Das dritte Unterprogramm lautet **Update_Display**. Der Befehl **LOOKUP** in diesem Unterprogramm enthält eine Tabelle mit sechs Bitmustern, die für die Darstellung des Kreismusters auf der 7-Segment Anzeige benötigt werden. Da bei jedem Unterprogrammaufruf die Variable **index** um 1 erhöht wird, wird das folgende Bitmuster aus der Tabelle an **OUTH** übergeben. Die Lookup Tabelle enthält nur Einträge für die **index** Werte von 0 bis 5. Was passiert, wenn der Indexwert 6 anzeigt? Der Lookup Befehl kann diesen Wert nicht interpretieren, deshalb lässt sich mit einer **IF…THEN** Anweisung der nicht definierte Zustand beheben. Die Befehlszeile **IF index = 6 THEN index = 0** setzt den Wert von **index** auf 0, wenn er 6 anzeigt. Damit wird erreicht, dass sich die Folge der Bitmuster in **OUTH** ständig wiederholen. Das wiederum lässt die 7-Segment Anzeige das Kreismuster ständig wiederholen.

```
Update_Display:
  IF index = 6 THEN index = 0
```

```
  '                BAFG.CDE
  LOOKUP index, [ %01000000,
                  %10000000,
                  %00000100,
                  %00000010,
                  %00000001,
                  %00100000 ], OUTH
  index = index + 1
RETURN
```

Du bist dran – Hard- und Software des Messgerätes justieren

Es bestehen zwei Möglichkeiten, die Empfindlichkeit des Messgerätes zu verändern. Ein Weg führt über die Software. Die 7-Segment Anzeige wird 10-mal langsamer zyklisch durchlaufen, wenn die Zeitvariable im **`Delay`** Unterprogramm vorher mit 10 multipliziert wurde, und sie wird zyklisch doppelt so schnell durchlaufen, wenn die gleiche Variable durch 2 geteilt wird.

√ Multipliziere die **`time`** Variable im Programm LightMeter.bs2 mit 10. Dies geschieht durch die Abänderung der Befehlszeile

```
PAUSE time
```

in

```
PAUSE time * 10
```

im Unterprogramm **`Delay`**.

√ Führe das veränderte Programm aus und überprüfe, ob die zyklische Darstellung auf der 7-Segment LED Anzeige 10-mal langsamer als vorher abläuft.

√ Wähle in einem zweiten Schritt andere Multiplikations- bzw. Divisionsfaktoren für die **`time`** Variable, wie zum Beispiel 5 oder 20 oder den Quotienten 2.

Das zyklische Aufleuchten der Dioden lässt sich auch dadurch um den Faktor 10 verlangsamen, indem man den 0,01 µF Kondensators gegen einen von 0,1 µF austauscht. Erinner dich daran, dass ein Kondensator mit einer 10-fach höheren Kapazität zu einer 10-mal höheren Zeitkonstante führt.

√ Tausche den 0,01 µF Kondensator gegen einen mit 0,1 µF aus.

√ Führe das Programm aus und überprüfe, ob der vorhergesagte Effekt eintritt.

Was ist besser, Anpassung der Hard- oder Software? Dein Bestreben sollte immer sein, das Beste aus den beiden Welten zu benutzen. Wähle einen Kondensator, der über einen großen Lichtbereich möglichst genaue Messwerte liefert. Hast du die besten Hardwarekomponenten zusammengestellt, justiere über die Software das Lichtmessgerät für Innen- und Außenmessungen. Das erfordert viele Tests und Verbesserungen, aber dies ist genau das Vorgehen in einem Entwicklungsprozess bei Einführung eines neuen Produkts.

ZUSAMMENFASSUNG

In diesem Kapitel wurde über Lichtmessungen mit einem LDR ein weiterer Weg aufgezeigt, wie sich der Befehl **RCTIME** einsetzen lässt. Wie das Potentiometer ist auch der Fotowiderstand ein variabler Widerstand. Anders als beim Potentiometer, dessen Widerstandswert sich mit der Position des Schleifers ändert, ändert der Fotowiderstand seinen Widerstandswert bei unterschiedlichen Lichtintensitäten. Stamp Plot Lite stellte Lichtmessungen graphisch dar und es wurde eine neue Methode kennengelernt, Daten aufzunehmen und zu interpretieren. Die Befehle **WRITE** und **READ** lernten wir als Datentransferbefehle zum BASIC Stamp EEPROM Speicher kennen: mit **WRITE** wurden Daten in den Speicher geschrieben, mit **READ** aus ihm heraus gelesen. In einer Anwendung haben wir das EEPROM als nicht flüchtigen Speicher für die Aufnahme von Entladezeiten benutzt. In der letzten Übung dieses Kapitels entwickelten wir eine Anwendung für ein Lichtmessgerät. Die Anwendung enthielt Unterprogramme, die drei unterschiedliche Aufgaben innerhalb der Gesamtanwendung übernahmen.

Verständnisfragen

1. Welche physikalischen Größen lassen sich mit Sensoren erfassen?
2. Wie nennt sich die chemische Substanz, die im Fotowiderstand verarbeitet wird?
3. Welche Gemeinsamkeiten hat ein Fotowiderstand mit einem Potentiometer? Welche Unterschiede bestehen zwischen ihnen?
4. Wofür steht das Akronym EEPROM?
5. Wie viele Byte kann das EEPROM in einem BASIC Stamp Modul speichern? Wie viele Bit können dort abgespeichert werden?
6. Mit welchem Befehl lässt sich ein Wert in einem EEPROM abspeichern? Mit welchem Befehl kann ein Wert aus einem EEPROM ausgelesen werden? Welcher Befehl benötigt einen Variablenaufruf?
7. Was versteht man unter einer Marke?
8. Was ist ein Unterprogramm?
9. Mit welchem Befehl wird ein Unterprogramm aufgerufen? Welcher Befehl beendet ein Unterprogramm?

Übungen

1. Zeichne das Schaltbild einer RC-Schaltung mit einem Fotowiderstand, der mit P5 verbunden ist.
2. Änder TestPhotoresistor.bs2 so ab, dass es mit einer Schaltung zusammenarbeitet, die mit P5 statt mit P2 verbunden ist.

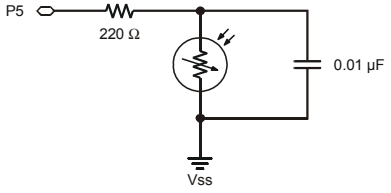
3. Erkläre, wie LightMeter.bs2 verändert werden muss, damit die zyklische Darstellung auf der 7-Segment LED Anzeige entgegen dem Uhrzeigersinn abläuft.

Projekt

1. In einem der vorherigen Kapitel hast du mit einem Taster eine LED zum Blinken gebracht. Statt des Tasters benutze jetzt einen Fotowiderstand, um die LED zum Blinken zu bringen, wenn der LDR beschattet wird. Hinweis: Benutze eine **`IF…THEN`** Anweisung und die Operatoren größer als/kleiner als, um zu entscheiden, ob die **`time`** Messung über oder unter einem bestimmten Wert liegt. Der Operator **`>`** steht für größer als, der Operator **`<`** für kleiner als.

Lösungen

Q1. Druck, Position, Drehung, Temperatur, Rauch, Vibration, Schlag, Licht und viele weitere Größen: Feuchtigkeit, Anziehungskraft, Durchflussrate, u.v.a.m.
Q2. Cadmiumsulfid (CdS).
Q3. Beide Bauteile sind veränderbare Widerstände. Beim Fotowiderstand geschieht dies über unterschiedlichen Lichteinfall, beim Potentiometer über Positionsveränderung des Schleifkontaktes.
Q4. Electrically Erasable Programmable Read-Only Memory.
Q5. 2048 bytes. 2048 x 8 = 16,384 bits.
Q6. Um einen Wert zu speichern – **`WRITE`**
Um einen Wert auszulesen – **`READ`**
Der **`READ`** Befehl arbeitet mit einem Variablenaufruf.
Q7. Eine Marke ist ein Name, der als Platzhalter in einem PBASIC Programm eingesetzt werden kann.
Q8. Ein Unterprogramm ist ein kleines Programm, das eine bestimmte Aufgabe ausführt.
Q9. Aufruf: **`GOSUB`**; Ende: **`RETURN`**
E1. Schaltungsaufbau aus Abbildung 7-2 auf Seite 203, P2 wird abgeändert zu P5.



E2. Die notwendigen Änderungen sind ähnlich denen von Seite 203.

```
DO
  HIGH 5
  PAUSE 100
  RCTIME 5, 1, time
  DEBUG HOME, "time =  ", DEC5 time
LOOP
```

E3. Bei entgegengesetzter Drehrichtung müssen die Muster in umgekehrter Reihenfolge dargestellt werden. Dies kann dadurch erreicht werden, dass die Muster in der **LOOKUP** Anweisung entsprechend vertauscht werden.

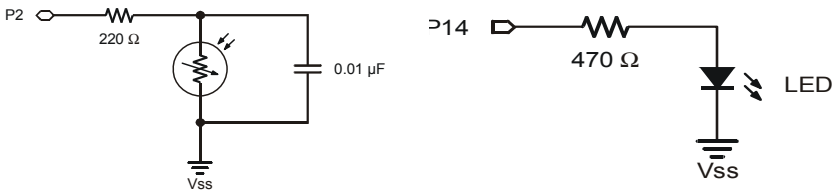
Lösung 1

```
Update_Display:
IF index = 6 THEN index = 0
'                BAFG.CDE
LOOKUP index, [ %01000000,
                %10000000,
                %00000100,
                %00000010,
                %00000001,
                %00100000 ], OUTH
index = index + 1
RETURN
```

Lösung 2

```
Update_Display:
'                  BAFG.CDE
  LOOKUP index, [ %01000000,
                  %10000000,
                  %00000100,
                  %00000010,
                  %00000001,
                  %00100000 ], OUTH
  IF (index = 0) THEN
    index = 5
  ELSE
    index = index - 1
  ENDIF
RETURN
```

P1. Fotowiderstand aus Abbildung 7-2, S.203; LED aus Abbildung 2-11, S.50.



Mit Hilfe einer **IF…THEN** Anweisung, die überprüft, ob der Fotowiderstand oberhalb eines Schwellenwertes liegt, kann dieses Problem gelöst werden. Ist dies der Fall, leuchtet die LED. Der Schwellenwert kann über die Ausführung des Programms TestPhotoresistor.bs2 und Beobachtung der gelesenen Werte gefunden werden. Denke an die unterschiedlichen Messwerte bei einem beschatteten und unbeschatteten LDR. Wähle einen mittleren Wert als Schwellenwert. In der dargestellten Lösung wurde der Schwellenwert in einer Konstanten **Dark** abgelegt, um das Programm leichter abändern zu können.

```
' What's a Microcontroller - Ch07Prj01_PhotoresistorFlasher.bs2
' Make LED on P14 flash whenever a shadow is cast over
' the photoresistor.  Change "Dark" constant for your conditions.

' {$STAMP BS2}
' {$PBASIC 2.5}

Dark    CON     25
time    VAR     Word

DO

  HIGH 2                        ' Read photoresistor with RCTIME
  PAUSE 100
  RCTIME 2, 1, time

  DEBUG HOME, "time =  ", DEC5 time ' Print value to Debug Terminal

  IF (time > Dark) THEN         ' Compare reading to known dark value
    HIGH 14                     ' Blink LED on pin P14
    PAUSE 100
    LOW 14
    PAUSE 100
  ENDIF

LOOP
```

Weitere Untersuchung

"*Applied Sensors*", Student Guide, Version 2.0, Parallax Inc., 2003

In diesem Buch werden neben weiteren Anwendungsschaltungen mit Sensoren vertiefende Betrachtungen zu Lichtmessungen mit Fotodioden, wissenschaftliche Einheiten und die Mathematik angestellt.

"*Industrial Control*", Student Guide, Version 2.0, Parallax Inc., 2002

Stamp Plot Lite wurde im Zusammenspiel mit der Entstehung dieses Buch entwickelt, um die grundlegenden Techniken darzustellen, die bei der industriellen Prozessüberwachung angewendet werden.

Kapitel 8: Frequenzen und Töne

EIN LEBEN MIT ELEKTRONISCHEN SIGNALTÖNEN

Im Verlauf eines normalen Arbeitstages gibt es viele Gelegenheiten, Signaltöne wahrzunehmen: die Mikrowelle meldet sich mit einem Signalton, sobald die eingestellte Zeit abgelaufen ist, ein Mobilphone erzeugt verschiedene Signaltöne, um anzuzeigen, ob eine SMS angekommen ist oder ein Anrufer uns zu sprechen wünscht, ein Warnsignal wird im Auto ausgegeben, wenn man sich nicht angeschnallt hat und im Kaufhaus zeigt ein Signalton an, dass der Barcode der einzuscannenden Ware erkannt worden ist. Und vielleicht wurdest du ja gerade heute morgen von einem Signalton des Weckers geweckt.

MIKROCONTROLLER, LAUTSPRECHER, SIGNALTÖNE UND HIGH/LOW SIGNALE

Nahezu alle elektronischen Signaltöne werden von Mikrocontrollern, die mit einem Lautsprecher verbunden sind, aufgebaut. Der Controller erzeugt den Ton durch das schnelle Aussenden von HIGH/LOW Signalen. Die Anzahl der gesendeten HIGH/LOW Signale pro Sekunde bezeichnen wir als Frequenz, sie bestimmt die Tonhöhe des Signaltons. Jedesmal, wenn sich ein HIGH/LOW Signal wiederholt, sprechen wir von einer Schwingung, die Zeit, die dabei vergeht, wird als Schwingungsdauer bezeichnet. Die Anzahl der Schwingungen pro Sekunde wird in der Einheit Hertz angegeben und mit Hz abgekürzt. Eine der am häufigsten eingesetzten Warnfrequenzen bei Geräten liegt bei 2 kHz. Das heißt, dass sich die HIGH/LOW Signale 2000 mal pro Sekunden wiederholen.

Der Piezoschallwandler

In dieser Übung werden verschiedene Signale an einen einfachen, kleinen und preisgünstigen Piezoschallwandler übertragen. Sein Schaltbild und die Bauteilzeichnung zeigt Abbildung 8-1.

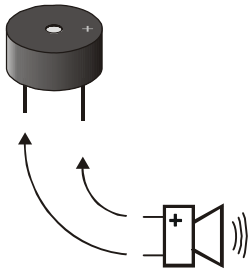


Abbildung 8-1
Piezoelektrischer Schallwandler Schaltsymbol und Bauteildarstellung

Ein piezoelektrischer Schallwandler wird im allgemeinen auch als Summer oder Alarmgeber angesprochen.

ÜBUNG 1: AUFBAU UND TEST EINER SCHALTUNG MIT SCHALLWANDLER

In dieser Übung wird eine Schaltung mit einem Summer aufgebaut und getestet

Bauteile Schallwandler

1 Piezoelektrischer Schallwandler
2 Verbindungsdrähte

Schaltungsaufbau mit Piezoschallwandler

Der negativ markierte Anschluss des Alarmgebers wird mit Vss verbunden, der positive mit einem I/O Pin. Die BASIC Stamp programmieren wir so, dass HIGH/LOW Signale an den positiv markierten Anschluss des Wandlers übertragen werden.

√ Baue die Schaltung aus Abbildung 8-2 auf.

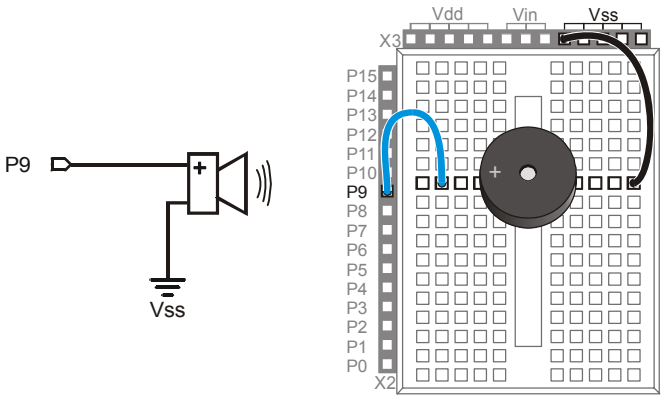


Abbildung 8-2
Schaltung mit piezoelektrischem Schallwandler Schaltungsaufbau und Verdrahtungsplan

Arbeitsweise der Schaltung mit Piezoschallwandler

Eine angeschlagene Gitarrensaite verursacht Luftdruckänderungen, die von unserem Ohr als Ton wahrgenommen werden kann. Je schneller diese Änderungen stattfinden, um so höher ist der wahrgenommene Ton, je langsamer die Luftdruckänderungen stattfinden, um so tiefer klingt der Ton. In einem Alarmgebergehäuse befindet sich ein sogenanntes Piezoelement. Werden HIGH/LOW Signale an den positiven Wandleranschluss gelegt, fängt der piezoelektrische Kristall zu schwingen an und bringt die Luft vor ihm ebenfalls zum Schwingen. Die dadurch entstehenden Luftdruckänderungen werden, ähnlich wie bei einer angeschlagenen Gitarrensaite, vom Ohr als Ton wahrgenommen.

Programm zur Steuerung eines Schallwandlers

Der Befehl **`FREQOUT`** ist eine einfache Lösung, HIGH/LOW Signale an einen Wandler zu übertragen und Töne zu erzeugen. Nach dem *BASIC Stamp Manual* lautet die Syntax für diesen Befehl:

FREQOUT ***Pin, Duration, Freq1 {, Freq2}***

Wie mit fast allen anderen Befehlen, die wir in diesem Buch verwenden, gibt ***`Pin`*** den BASIC Stamp I/O Pin an, der angesprochen werden soll. Das Argument ***`Duration`*** gibt in der Zeiteinheit Millisekunden an, wie lange der über **`FREQOUT`** ausgegebene Ton erklingen soll. Das Argument ***`Freq1`*** legt die Frequenz des Tones in Hertz fest. Das optionale Argument ***`Freq2`*** ermöglicht das Mischen von Frequenzen.

Der folgende Befehlsaufruf zeigt, wie man einen Ton von 2 kHz über eine Dauer von 1,5s an I/O Pin P9 überträgt:

```
FREQOUT 9, 1500, 2000
```

Programmbeispiel: TestPiezoWithFreqout.bs2

Dieses Programm sended 1,5 Sekunden lang einen 2kHz Ton an einen Schallwandler, der am I/O Pin P9 angeschlossen ist. Auf dem Debug Terminal kann verfolgt werden, wann der Alarmgeber den Ton ausgibt und wann der Ton stoppt.

- √ Führe das Programm TestPiezoWithFreqout.bs2 aus.
- √ Überzeuge dich, dass der Wandler, während auf dem Debug Terminal die Mitteilung „Tone sending ...“ erscheint, einen klaren Ton ausgibt.

```
' What's a Microcontroller - TestPiezoWithFreqout.bs2
' Send a tone to the piezo speaker using the FREQOUT command.

'{$STAMP BS2}
'{$PBASIC 2.5}

DEBUG "Tone sending...", CR

FREQOUT 9, 1500, 2000

DEBUG "Tone done."
```

Du bist dran –Frequenz und Tondauer anpassen

- √ Speicher TestPiezoWithFreqout.bs2 unter einem andern Namen ab.
- √ Setze für ***Duration*** und ***Freq1*** verschiedene andere Werte ein.
- √ Nach jeder Änderung starte das Programm erneut und nimm die Unterschiede wahr.
- √ Wird das Argument ***Freq1*** größer, wird dann der Ton höher oder niedriger? Gib Werte von 1500, 2000, 2500 und 3000 ein und beantworte die Frage.

ÜBUNG 2: SPIELGERÄUSCHE ERZEUGEN

Viele Spiele enthalten Mikrocontroller, die nur für die Ausgabe von Spielgeräuschen zuständig sind. Sie bestehen aus sich schnell ändernden Frequenzmustern. Einige interessante Effekt lassen sich aber auch dadurch erzielen, dass man zwei unterschiedliche Töne über das Argument ***Freq2*** im Befehl **FREQOUT** mischt. Diese Übung vermittelt beide Techniken.

Programmierung von Spielgeräuschen

Spielgeräusche und Gerätetöne bauen sich aus drei unterschiedlichen Komponenten auf:

1. Pause
2. Tondauer
3. Frequenz

Die Pause ist die Zeit, die zwischen zwei Tönen vergeht; über den Befehl **PAUSE** wird die Pausenzeit festgelegt. Die Tondauer meint die Zeit, über die ein Ton zu hören ist; sie wird über das Argument ***Duration*** im Befehl **FREQOUT** gesetzt. Die Frequenz bestimmt die Tonhöhe. Je höher die Frequenz, desto höher klingt der Ton, je niedriger die Frequenz, desto tiefer klingt er. Die Einstellung erfolgt über das Argument ***Freq1*** im Befehl **FREQOUT**.

Programmbeispiel: ActionTones.bs2

ActionTones.bs2 demonstriert zwei unterschiedliche Kombinationen der drei genannten Größen Pause, Tondauer und Frequenz. Die erste Tonsequenz klingt eher wie eine elektronische Alarmglocke. Die zweite erinnert vielleicht an die Sprache einer bekannte Science Fiction Roboterfigur und die dritte könnte Ähnlichkeit mit Soundeffekten aus alten Videospielen haben.

√ Führe das Programm ActionTones.bs2 aus.

```
' What's a Microcontroller - ActionTones.bs2
' Demonstrate how different combinations of pause, duration, and frequency
' can be used to make sound effects.

'{$STAMP BS2}
'{$PBASIC 2.5}

duration       VAR     Word
frequency      VAR     Word

DEBUG "Alarm...", CR
   PAUSE 100
   FREQOUT 9, 500, 1500
   PAUSE 500
   FREQOUT 9, 500, 1500
   PAUSE 500
   FREQOUT 9, 500, 1500
   PAUSE 500
   FREQOUT 9, 500, 1500
```

```
   PAUSE 500

DEBUG "Robot reply...", CR
   PAUSE 100
   FREQOUT 9, 100, 2800
   FREQOUT 9, 200, 2400
   FREQOUT 9, 140, 4200
   FREQOUT 9,  30, 2000
   PAUSE 500

DEBUG "Hyperspace...", CR
  PAUSE 100
  FOR duration = 15 TO 1 STEP 1
    FOR frequency = 2000 TO 2500 STEP 20
      FREQOUT 9, duration, frequency
    NEXT
  NEXT

DEBUG "Done", CR

END
```

Arbeitsweise von ActionTones.bs2

Die “Alarm”-Routine hört sich wie ein Wecker an. Es werden Töne mit einer Frequenz von 1,5 kHz, einer Dauer von 0,5 Sekunden und einer Pause von 0,5 Sekunden ausgegeben. Die “Robot reply” Routine arbeitet mit unterschiedlichen Frequenzen und kurzen Tonlängen.

Die “Hyperspace” Routine verwendet keine Pausen, variiert aber sowohl die Tonlänge als auch die Frequenz. Durch die Verwendung von **`FOR…NEXT`** Schleifen, die für eine schnelle Änderung von Frequenz und Tonlänge verantwortlich sind, enstehen einige interessante Soundeffekte. Die Wirkungsweise der geschachtelten **`FOR…NEXT`** Schleifen soll jetzt besprochen werden. Die Variable **`duration`** startet mit dem Wert 15, die folgende **`frequency`** Schleife überträgt dann die Frequenzen 2000, 2020, 2040 bis 2500 an den Piezoschallwandler. Ist die innere Schleife abgearbeitet, wird in der äußeren **`duration`** Schleife die Laufvariable um eins erniedrigt und die **`frequency`** Schleife erneut durchlaufen; insgesamt wiederholt sich dieser Vorgang 15 mal.

```
FOR duration = 15 TO 1
  FOR frequency = 2000 TO 2500 STEP 15
    FREQOUT 9, duration, frequency
  NEXT
NEXT
```

Programmbeispiel: NestedLoops.bs2

Um besser verstehen zu können, wie geschachtelte **FOR…NEXT** Schleifen arbeiten, enthält NestedLoops.bs2 den **DEBUG** Befehl, der die Indexwerte einer einfachen geschachtelten Schleife anzeigt.

- √ Führe das folgende Programm aus: NestedLoops.bs2.
- √ Beobachte die Ausgabe auf dem Debug Terminal und überprüfe, wie die Argumente von Tondauer (duration) und Frequenz sich bei jedem Schleifendurchlauf ändern.

```
' What's a Microcontroller - NestedLoops.bs2
' Demonstrate how the nested loop in ActionTones.bs2 works.

'{$STAMP BS2}
'{$PBASIC 2.5}

duration       VAR     Word
frequency      VAR     Word

DEBUG "Duration   Frequency",CR,
      "--------   ---------",CR

FOR duration = 4000 to 1000 step 1000
   FOR frequency = 1000 to 3000 step 500
       DEBUG "   " , DEC5 duration,
             "     " , DEC5 frequency, CR
       FREQOUT 9, duration, frequency
   NEXT
   DEBUG CR
NEXT

END
```

Du bist dran – Weitere Soundeffekte

Es gibt eine unbeschränkte Anzahl von Möglichkeiten ActionTones.bs2 zu verändern, und unterschiedliche Soundkombinationen zu erhalten. Das folgende Beispiel zeigt eine Änderung der "Hyperspace" Routine:

```
DEBUG "Hyperspace jump...", CR
  FOR duration = 15 TO 1 STEP 3
    FOR frequency = 2000 TO 2500 STEP 15
      FREQOUT 9, duration, frequency
    NEXT
  NEXT
```

```
FOR duration = 1 TO 36 STEP 3
  FOR frequency = 2500 TO 2000 STEP 15
    FREQOUT 9, duration, frequency
  NEXT
NEXT
```

√ Speicher das Beispielprogramm unter dem Namen ActionTonesYourTurn.bs2 ab.

√ Viel Spaß mit dieser Modifikation und weiteren, die nach eigenen Vorstellungen entwickelt werden.

Zwei Frequenzen mischen

Es lassen sich auch zwei Frequenzen gleichzeitig ausgeben. Im Audiobereich spricht man von „mixing“ oder mischen”. Erinner dich an die Syntax des Befehls **`FREQOUT,`** die wir in Übung 1 angesprochen haben:

FREQOUT ***Pin, Duration, Freq1 {, Freq2}***

Über das optionale Argument ***`Freq2`*** im Befehl **`FREQOUT`** lassen sich zwei Frequenzen mischen. Für die Frequenzen 2 kHz und 3 kHz sieht ein Programmaufruf wie folgt aus:

```
FREQOUT 9, 1000, 2000, 3000
```

Jeder Ton eines Tonwahltastenfeldes setzt sich ebenfalls aus zwei Frequenzen zusammen. In der Fernmeldetechnik sprechen wir von DTMF (Dual Tone Multi Frequency). Über den PBASIC Befehl **`DTMFOUT`** lassen sich solche Telefontöne erzeugen. Wenn du mehr darüber erfahren möchtest, wie man eine Telefonanwahl durchführt, findest du weitere Informationen unter **`DTMFOUT`** im BASIC Stamp Manual.

Programmbeispiel: MixingTones.bs2

Dieses Beispielprogramm zeigt was passiert, wenn ein 2 kHz und ein 3 kHz Ton miteinander gemischt werden. Liegen die beiden Frequenzen der zu mischenden Töne sehr eng beieinander, tritt ein interessantes Phänomen auf, das die Physiker Schwebung nennen. Werden zum Beispiel ein 2000 Hz und ein 2001 Hz Ton überlagert, wird der resultierende Ton mit einer Frequenz von 1 Hz an- und abschwellen. Bei der Überlagerung zweier Töne von 2000 Hz und 2002 Hz ergibt sich entsprechend eine Schwebung von 2 Hz.

Wir sprechen von einer Schwebung, wenn die Frequenzen zweier Töne sehr nah beieinander liegen und der resultierende Ton an- und abschwillt. Die Frequenz, mit der ein Ton an- und abschwillt, wird durch die Differenz der beiden sich überlagernden Frequenzen bestimmt. Bei einer Differenz von 1 Hz schwillt der Ton ebenfalls mit einer Frequenz von 1 Hz an und ab. Für alle anderen Frequenzen gilt die Aussage entsprechend.

Die vom Piezoschallwandler verursachten Luftdruckschwankungen bezeichnen wir als akustische Wellen, es sind sogenannte Longitudinalwellen. Wird der Ton sehr laut, dann haben sich die beiden Wellen nach dem Superpositionsprinzip zu einem maximalen Wert überlagert. Der leiseste Ton kommt dadurch zustande, dass sich bei der Überlagerung der beiden Wellenzüge die Wellen gegenseitig durch Interferenz auslöschen.

√ Führe das Programm MixingTones.bs2 aus.
√ Beobachte das Debug Terminal während die Töne abgespielt werden und beachte die unterschiedlichen Effekte, die bei der Überlagerung der unterschiedlichen Töne auftreten.


```
' What's a Microcontroller - MixingTones.bs2
' Demonstrate some of the things that happen when you mix two tones.

'{$STAMP BS2}
'{$PBASIC 2.5}

DEBUG "Frequency = 2000", CR
FREQOUT 9, 4000, 2000

DEBUG "Frequency = 3000", CR
FREQOUT 9, 4000, 3000

DEBUG "Frequency = 2000 + 3000", CR
FREQOUT 9, 4000, 2000, 3000

DEBUG "Frequency = 2000 + 2001", CR
FREQOUT 9, 4000, 2000, 2001

DEBUG "Frequency = 2000 + 2002", CR
FREQOUT 9, 4000, 2000, 2002

DEBUG "Frequency = 2000 + 2003", CR
FREQOUT 9, 4000, 2000, 2003

DEBUG "Frequency = 2000 + 2005", CR
FREQOUT 9, 4000, 2000, 2005

DEBUG "Frequency = 2000 + 2010", CR
FREQOUT 9, 4000, 2000, 2010

DEBUG "Done", CR
```

```
END
```

Du bist dran – Speicherplatzsparend programmieren

MixingTones.bs2 hat uns gezeigt, was passiert, wenn zwei akustische Wellen unterschiedlicher Frequenz mit Hilfe des Arguments ***Freq2*** im Befehl **FREQOUT** überlagert werden. Dieses Programmierverfahren ist allerdings sehr unwirtschaftlich.

√ Änder das Programm MixingTones.bs2 so ab, dass mit Hilfe einer Schleife das Argument ***Freq2*** die Werte 2001 bis 2005 durchläuft. Deklariere dazu eine Variable vom Typ Word.

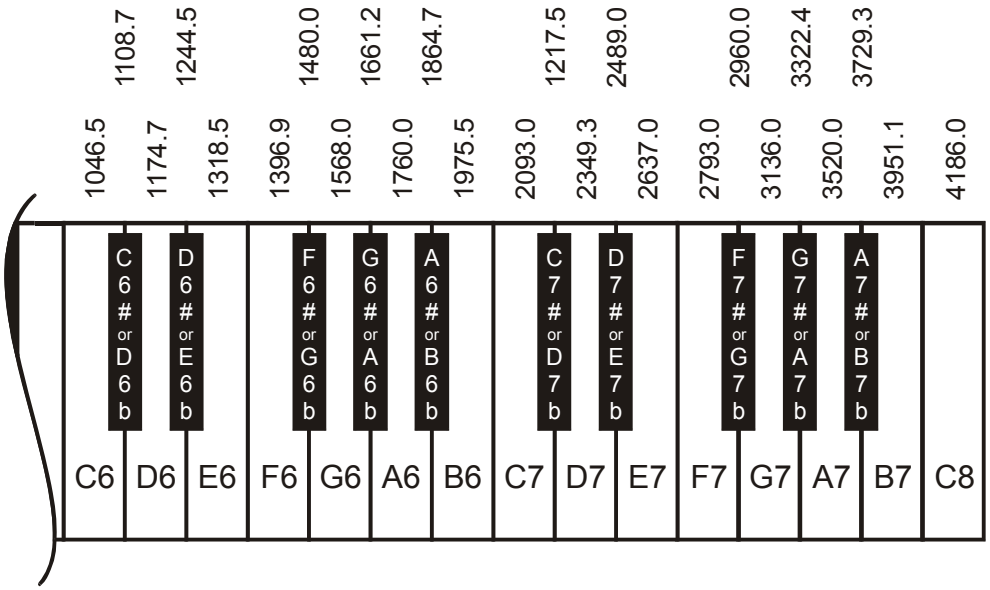
ÜBUNG 3: NOTEN UND EINFACHE SONGS PROGRAMMIEREN

Abbildung 8-3 zeigt den Ausschnitt von 25 Tasten einer Klaviertastatur. Dort sind ebenfalls die Frequenzen angegeben, mit der jede Saite schwingt, sobald sie angeschlagen wird. Die Tasten und die entsprechenden Noten sind mit C6 bis C8 bezeichnet. Jeweils 12 Tasten, 8 weiße und 4 schwarze, bilden eine Gruppe und umfassen eine Oktave. Die Notenfolge wiederholt sich nach jeweils 12 Tasten. Noten gleicher Bezeichnung haben dieselbe Frequenz; mit jeder Oktave verdoppelt sich die Frequenz. C7 hat beispielsweise eine doppelt so hohe Frequenz wie C6, C8 eine doppelt so hohe wie C7. Umgekehrt ist die Frequenz von A6 halb so groß wie die von A7.

Internetsuche unter dem Stichwort – "Tonleiter": Gibt man das Wort "Tonleiter" in eine Suchmaschine wie Google oder Yahoo ein, erhält man umfangreiche Informationen zu diesem Stichwort. Die westliche Musik basiert auf 12 Notenstufen pro Oktave. Andere Kulturen kommen auf 2 bis 35 Noten pro Oktave.

Beim Einsingen solmisieren Sänger häufig das bekannte "Do Re Mi Fa So La Ti Do", eine Ganztonfolge, die auf dem Klavier mit den weißen Tasten nachgespielt werden kann; die schwarzen Tasten geben Halbtöne wieder. Die schwarze Taste zwischen den Tasten C und D ist zum Beispiel für den Ton Cis($C^{\#}$) oder Des(D^{b}) zuständig. Wenn du mehr über die hier nur sehr kurz angeschnittene Musiktheorie erfahren möchtest, wende dich bitte an deinen Musiklehrer.

Abbildung 8-3: Klaviertasten (Ausschnitt) und die zugehörigen Frequenzen



Die Stimmung: Die Stimmung der Tastatur in Abbildung 8-3 bezeichnen wir als gleichmäßig oder wohl temperiert. Die Frequenzen werden von einer Referenznote abgeleitet und dann mit $2^{(n/12)}$ für n = 1, 2, 3, etc. multipliziert. Um die Frequenz von A6# zu bestimmen, wird die Frequenz von A6 mit $2^{(1/12)}$ multipliziert. Multiplikation mit $2^{(2/12)}$ ergibt die Frequenz von B6 usw.. Im folgenden Beispiel wird die Frequenz von B6 berechnet, dabei wird A6 als Referenzfrequenz benutzt:

Die Frequenz von A6 ist mit 1760 Hz angegeben.

$2^{(2/12)} = 1.1224$

1760 X 1.224 = 1975.5

1975.5 Hz ist die Frequenz von B6

Notenprogrammierung

Der Befehl **`FREQOUT`** ist zum Schreiben von Noten gut geeignet. Um über den an einer BASIC Stamp angeschlossenen Piezoschallwandler Musik abzuspielen, gilt es eine Reihe von Regeln zu beachten, die genau so auch beim Spielen eines Musikinstrumentes eingehalten werden müssen. Diese Regeln beziehen sich auf die Größen Frequenz,

Tondauer und Pause. Im folgenden Beispielprogramm werden einige Noten, jede mit einer Tonlänge von einer halben Sekunde, über den Schallwandler abgespielt.

Programmbeispiel: DoReMiFaSolLaTiDo.bs2

√ Für das Programm DoReMiFaSolLaTiDo.bs2 aus.

```
' What's a Microcontroller - DoReMiFaSolLaTiDo.bs2
' Send an octave of half second tones using a piezoelectric speaker.

'{$STAMP BS2}
'{$PBASIC 2.5}

'Solfege            Tone                    Note

DEBUG "Do...", CR:  FREQOUT 9,500,1047     ' C6

DEBUG "Re...", CR:  FREQOUT 9,500,1175     ' D6

DEBUG "Mi...", CR:  FREQOUT 9,500,1319     ' E6

DEBUG "Fa...", CR:  FREQOUT 9,500,1396     ' F6

DEBUG "Sol..", CR:  FREQOUT 9,500,1568     ' G6

DEBUG "La...", CR:  FREQOUT 9,500,1760     ' A6

DEBUG "Ti...", CR:  FREQOUT 9,500,1976     ' B6

DEBUG "Do...", CR:  FREQOUT 9,500,2093     ' C7

END
```

Du bist dran – Notenfolgen speichern und zurückladen

√ Orientiere dich an den Frequenzen aus Abbildung 8-3 und füge in DoReMiFaSolLaTiDo.bs2 die fünf fehlenden Halbtöne hinzu.

√ Änder das Programm so ab, dass es auch die folgende Oktave spielt. Hinweis: Erleichter dir die Arbeit dadurch, dass du den Operator **`* 2`** bei jedem ***`Freq1`*** Argument einsetzt. Beispiel: **`FREQOUT 9, 500, 1175 * 2`** ergibt D7, die Note D in der 7. Oktave.

Notenfolgen speichern und zurückladen

Das EEPROM auf dem BASIC Stamp Modul eignet sich gut zum Speichern von Noten. Wir wissen bereits, dass das mit Hilfe des Befehls **WRITE** erfolgt; einfacher geht es über die Direktive **DATA**. Die Syntax der **DATA** Direktive lautet:

{Symbol} DATA ***{Word} DataItem {, {Word} DataItem, … }***

Im folgenden Beispiel wird gezeigt, wie die **DATA** Direktive aufgerufen wird, um Zeichen, die entsprechenden Noten zugewiesen sind, zu speichern.

```
Notes DATA "C","C","G","G","A","A","G"
```

Mit dem **READ** Befehl wird auf die Zeichen zugegriffen. Der Buchstabe 'C' befindet sich an der Adresse **Notes + 0** ein zweiter Buchstabe 'C' an der Adresse **Notes + 1**. Der Buchstabe 'G' befindet sich bei **Notes + 2** und so weiter. Wenn zum Beispiel der letzte Buchstabe 'G' in eine Variable **noteLetter** vom Typ Byte geladen werden soll, lautet der Befehl:


```
READ Notes + 6, noteLetter
```

Mit Hilfe der **DATA** Direktive lassen sich auch ganze Zahlenlisten speichern. Frequenzangaben und Tondauern, die die BASIC Stamp zu jeder Note als weitere Informationen benötigt, müssen in Variablen vom Typ Word abgelegt werden, da ihre Werte im allgemeinen größer als 255 sind. Ein Beispiel zeigt die folgende Zeile:

```
Frequencies DATA Word 2093, Word 2093, Word 3136, Word 3136,
                 Word 3520, Word 3520, Word 3136
```

Jeder dieser Werte benötigt zwei Byte, deshalb unterscheidet sich der Zugriff auf Zahlen mit dem read Befehl von denen auf Zeichen. 2093 ist an der Adresse **Frequencies + 0**, 2093 aber bei **Frequencies + 2** zu finden. Die erste Zahl 3136 befindet sich bei **Frequencies + 4** und die zweite 3136 bei **Frequencies + 6**.

Die Werte in der Frequencies DATA Direktive korrespondieren mit den Noten in der **Notes DATA** Direktive.

In der folgenden **FOR…NEXT** Schleife werden die **Notes DATA** in einer Variablen **noteLetter** abgelegt und anschließend die **Frequencies DATA** in der Variablen **noteFreq**.

```
FOR index = 0 to 6
  READ Notes + index, noteLetter
  READ Frequencies + (index * 2), Word noteFreq
  DEBUG noteLetter, "  ", DEC noteFreq, CR
NEXT
```

Was bewirkt der Term (index * 2)? Jeder Wert, der in der **Frequencies DATA** Direktive abgespeichert wird, benötigt zwei Byte (WORD-Type), während jedes Zeichen der **Notes DATA** Direktive mit einem Byte auskommt. Der **index** Wert erhöht sich bei jedem **FOR…NEXT** Schleifendurchlauf um 1. Das ist richtig für den Zugriff auf die Notenzeichen über den Befehl **READ Notes + index, noteLetter**. Für den Zugriff auf die **Frequencies** Liste muss die Variable **index** aber jedesmal um 2 erhöht werden. Dies wird in der Befehlszeile **READ Frequencies + (index * 2), Word noteFreq** berücksichtigt.

Im folgenden Beispielprogramm werden Noten und Notenlängen über **DATA** abgespeichert und mit Hilfe des Befehls **FREQOUT** jede Note mit ihrer spezifischen Frequenz- und Notenlängenangabe abgespielt. Das Ergebnis sind die ersten Takte aus dem Kinderlied Twinkle Twinkle Little Star.

Das Alphabetlied, das insbesondere von angelsächsischen Kindern gesungen wird, um sich an die Buchstabenreihenfolge des “ABC” zu erinnern, basiert auf der gleichen Notenfolge wie Twinkle Twinkle Little Star.

Programmbeispiel: TwinkleTwinkle.bs2

Dieses Beispielprogramm zeigt, wie die **DATA** Direktive zum Abspeichern von Listen aufgerufen und wie mit dem Befehl **READ** auf die Werte in der Liste zugegriffen wird.

- √ Führe das Programm TwinkleTwinkle.bs2 aus.
- √ Überzeuge dich davon, dass die Noten das Lied Twinkle Twinkle Little Star wiedergeben.
- √ Überprüfe auf dem Debug Terminal das das Programm wie vorgesehen arbeitet, indem es die Daten der drei **DATA** Direktiven jeweils anzeigt.

```
' What's a Microcontroller - TwinkleTwinkle.bs2
' Play the first seven notes from Twinkle Twinkle Little Star.

'{$STAMP BS2}
'{$PBASIC 2.5}

Notes            DATA  "C","C","G","G","A","A","G"

Frequencies      DATA  Word 2093, Word 2093, Word 3136, Word 3136,
                       Word 3520, Word 3520, Word 3136

Durations        DATA  Word 500, Word 500, Word 500, Word 500,
                       Word 500, Word 500, Word 1000

index            VAR   Nib
noteLetter       VAR   Byte
noteFreq         VAR   Word
noteDuration     VAR   Word

DEBUG    "Note  Duration  Frequency", CR,
         "----  --------  ---------", CR

FOR index = 0 TO 6

  READ Notes + index, noteLetter
  DEBUG "   ", noteLetter

  READ Durations + (index * 2), Word noteDuration
  DEBUG "      ", DEC4 noteDuration

  READ Frequencies + (index * 2), Word noteFreq
  DEBUG "       ", DEC4 noteFreq, CR

  FREQOUT 9, noteDuration, noteFreq

NEXT

END
```

Du bist dran – Noten hinzufügen und abspielen

Das letzte Programm spielte die ersten sieben Noten aus dem Lied Twinkle Twinkle Little Star. Die Worte lauten “Twin-kle twin-kle lit-tle star”. Das Lied setzt sich fort mit dem Text “How I won-der what you are”; die Noten sind F, F, E, E, D, D, C. Wie schon bei der ersten Zeile wird auch in der zweiten die letzte Note doppelt so lang gehalten wie die anderen. Um diese Zeile dem Programm TwinkleTwinkle.bs2 hinzuzufügen, müssen

alle **DATA** Direktiven entsprechend abgeändert werden. Vergiss dabei nicht, auch die **FOR…NEXT** Schleife so abzuändern, dass sie von 0 bis 13, statt von 0 bis 6 läuft.

√ Änder TwinkleTwinkle.bs2 so ab, dass es die ersten beiden Zeilen des Liedes abspielt.

ÜBUNG 4: MUSIK MIT DEM MIKROCONTROLLER

Notenlängen werden auf Notenblättern nicht in der Einheit Millisekunden angegeben, sondern als ganze, halbe, viertel, achtel, sechzehntel usw. Note beschrieben. Wie der Name schon sagt, dauert eine halbe Note nur halb so lang, wie eine ganze Note. Entsprechendes kann für alle anderen Notenlängen gesagt werden. Wie lang wird nun aber eine ganze Note gespielt? Dies wiederum hängt vom Musikstück ab und wird über die Tempoangabe dem Musiker mitgeteilt. So kann es passieren, dass eine ganze Note in dem einen Stück vier Sekunden, in einem andern Stück aber nur zwei Sekunden dauert.

Pausen sind die Zeiten zwischen Noten, in denen kein Ton gespielt wird. Die Länge einer Pause wird auch als ganze, halbe, viertel, achtel, sechzehntel und zweiundreißigstel Pause mit jeweils einem eigenen Symbol angegeben.

Verbesserungen beim Speichern und Laden von Musik

Es lassen sich Programme schreiben, die doppelt so viel Musik in der BASIC Stamp speichern, wenn mit dem Variablentyp Byte und nicht mit Word in den **DATA** Direktiven gearbeitet wird. Ein Programm kann aber auch so abgeändert werden, dass sich die Noten einfacher lesen lassen, indem man einige der gebräuchlichen Vereinbarungen für Noten und Notenlängen nutzt. In dieser Übung wollen wir zeigen, wie sich Informationen über Musik speichern lassen, die sich an das Konzept von Noten, Notenlängen und Pausen anlehnen. Dabei wird auch auf das Tempo eingegangen und in der folgenden Übung noch einmal aufgegriffen.

Die hier vorgestellten **DATA** Direktiven zeigen, wie Noten und Notenlängen für das folgende Beispielprogramm abgelegt werden. Beim Abspielen sollte das Lied „Bruder Jakob“ zu hören sein. In der **Notes DATA** Direktive werden nur die Noten gespeichert und über die Befehle **LOOKUP** und **LOOKDOWN** werden den einzelnen Noten die entsprechenden Frequenzen zugeordnet.

```
Notes           DATA    "C","D","E","C","C","D","E","C","E","F",
                        "G","E","F","G","Q"
```

```
Durations       DATA      4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
                          2,  4,  4,  2

WholeNote       CON     2000
```

Die erste Zahl in der Direktive **`Durations DATA`** teilt dem Programm mit, wie lang die erste Note in der Direktive **`Notes Data`** gespielt werden soll. Der zweite Durationeintrag bezieht sich auf die zweite Note und so weiter. Die Notenlängen werden nicht in Millisekunden angegeben. Statt dessen sind es deutlich kleinere Zahlen, die im Format Byte abgelegt werden können und nicht mehr das Prefix **`Word`** in der **`DATA`** Direktive benötigen. Vergleicht man die Angaben in Millisekunden mit den hier vorliegenden, fällt auf, dass die hier benutzte Technik sich enger an die Nomenklatur für Notenlängenangaben hält. Die folgende Liste gibt die Entsprechungen von Zahlenwert zur notierten Notenlänge wieder.

- 1 – ganze Note
- 2 – halbe Note
- 4 – viertel Note
- 8 – achtel Note
- 16 – sechzehntel Note
- 32 – zweiunddreißigstel Note

Nachdem jeder Wert aus der **`Durations DATA`** Direktive gelesen wurde, wird der anteilige **`WholeNote`** Wert berechnet und damit der für den Befehl **`FREQOUT`** benötigte ***`Duration`*** Wert bestimmt. Die Notenlänge jeder Note hängt vom Tempo des Liedes ab. Je höher das Tempo, desto kürzer wird die Note gespielt, während bei einem langsameren Tempo jede Noten länger gehalten werden muss. Da alle Notenlängen Bruchteile einer ganzen Note sind, lässt sich die Notenlänge einer ganzen Note als Maß für das Tempo einsetzen.

Was bedeutet das "Q" in `Notes DATA?` "Q" steht für ENDE; eine **`DO WHILE…LOOP`** Schleife wird beendet, sobald die Abbruchbedingung den Buchstaben „Q“ erkennt.

Wie lässt sich eine Pause aufrufen? Eine Pause kann zwischen Noten durch das Setzen von **`"P"`**eingefügt werden. Im Abschnitt Du bist dran ... werden die ersten Noten aus Beethovens 5. Symphony dargestellt, in der auch eine Pause vorkommt.

Wie lassen sich Halbtöne aufrufen? NotesAndDurations.bs2 ordnet in seiner Zuweisungstabelle den Halbtönen Werte zu. Werden die Noten mit kleinen Buchstaben aufgerufen, werden die um einen Halbton abgesenkten Töne gespielt. Soll zum Beispiel B^b gespielt werden, dann muss “b” statt “B” aufgerufen werden. Bedenke bitte, dass dieser Ton identisch mit Ais ($A^\#$) ist.

Programmbeispiel: NotesAndDurations.bs2

- √ Führe NotesAndDurations.bs2 aus.
- √ Wie klingt das Programm?

```
' What's a Microcontroller - NotesAndDurations.bs2
' Play the first few notes from Frere Jacques.

'{$STAMP BS2}
'{$PBASIC 2.5}

DEBUG "Program Running!"

Notes            DATA    "C","D","E","C","C","D","E","C","E","F",
                         "G","E","F","G","Q"

Durations        DATA      4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
                           2,  4,  4,  2

WholeNote        CON     2000

index            VAR     Byte
offset           VAR     Nib

noteLetter       VAR     Byte
noteFreq         VAR     Word
noteDuration     VAR     Word

DO UNTIL noteLetter = "Q"

  READ Notes + index, noteLetter

  LOOKDOWN noteLetter, [  "A",  "b",  "B",  "C",  "d",
                          "D",  "e",  "E",  "F",  "g",
                          "G",  "a",  "P",  "Q"     ], offset
```

```
  LOOKUP offset,       [ 1760, 1865, 1976, 2093, 2217,
                         2349, 2489, 2637, 2794, 2960,
                         3136, 3322,    0,    0     ], noteFreq

  READ Durations + index, noteDuration

  noteDuration = WholeNote / noteDuration

  FREQOUT 9, noteDuration, noteFreq

  index = index + 1

LOOP

END
```

Arbeitsweise von NotesAndDurations.bs2

Auf die Direktiven **`Notes DATA`** und **`Durations DATA`** sind wir weiter oben bereits eingegangen. Diese beiden und die Konstante **`WholeNote`** genügen, um die Informationen aller Noten, die das Programm abspielt, zu speichern.

Auf die Deklaration der fünf Variablen des Programms wird weiter unten eingegangen. Obwohl für den Datenzugriff keine **`FOR…NEXT`** Schleife mehr benötigt wird, muss über eine Indexvariable (**`index`**) darüber gewacht werden, welche Daten (**`DATA)`** in **`Notes`** und **`Durations`** jeweils eingelesen werden. Mit Hilfe der **`offset`** Variablen wird über die Befehle **`LOOKDOWN`** und **`LOOKUP`** ein bestimmter Wert ausgewählt. Die **`noteLetter`** Variable speichert ein Zeichen, auf das über den **`READ`** Befehl zugegriffen wurde. Die Befehle **`LOOKUP`** und **`LOOKDOWN`** wandeln das jeweilige Zeichen in einen Frequenzwert um. Dieser Wert wird in der Variablen **`noteFreq`** abgelegt und als Argument ***`Freq1`*** im Befehl **`FREQOUT`** aufgerufen. In die Variable **`noteDuration`** wird bei einem **`READ`** Befehl der ausgelesene Wert von **`Durations DATA`**. abgelegt. Über sie wird auch die ***`Duration`*** im Befehl **`FREQOUT`** berechnet.

```
index         VAR  Byte
offset        VAR  Nib

noteLetter    VAR  Byte
noteFreq      VAR  Word
noteDuration  VAR  Word
```

Die Hauptschleife wird solange durchlaufen, bis der Buchstabe ‘Q’ aus **`Notes DATA`**.ausgelesen wurde:

```
DO UNTIL noteLetter = "Q"
```

Mit dem **READ** Befehl wird ein Zeichen aus **Notes DATA** ausgelesen und in der Variablen **noteLetter** abgelegt. Die **noteLetter** Variable bestimmt über einen **LOOKDOWN** Befehl den Wert der Variablen **offset**. Erinnere dich daran, dass **offset** eine 1 enthält, wenn "b" erkannt wird, eine 2 wenn "B" und eine 3 wenn "C" erkannt wird. Mit diesem **offset** Wert wird dann über einen **LOOKUP** Befehl bestimmt, welchen Wert die Variable **noteFreq** annehmen soll. Bei einem **offset** von 1, nimmt **noteFreq** den Wert 1865, bei einem **offset** von 2, 1976 und bei einem **offset** von 3, 2093 an.

```
READ Notes + index, noteLetter

LOOKDOWN noteLetter, [  "A",  "b",  "B",  "C",  "d",
                        "D",  "e",  "E",  "F",  "g",
                        "G",  "a",  "P",  "Q" ], offset

LOOKUP offset,       [ 1760, 1865, 1976, 2093, 2217,
                       2349, 2489, 2637, 2794, 2960,
                       3136, 3322,    0,   0  ], noteFreq
```

Die Notenfrequenzen sind damit bestimmt, es fehlen nur noch die Tonlängen. Der Befehl **READ** benutzt den Wert von **index** um einen Zahlenwert aus **Durations DATA** in **noteDuration** abzulegen.

```
READ Durations + index, noteDuration
```

In einem weiteren Schritt wird **noteDuration** gleichgesetzt mit der Konstanten **WholeNote** dividiert durch **noteDuration**. Wird die Tonlänge nach Ausführung eines **READ** Befehls mit 4 ausgelesen, wird daraus 2000 ÷ 4 = 500. Ist **noteDuration** identisch 8, wird daraus 1500 ÷ 8 = 250.

```
noteDuration = WholeNote / noteDuration
```

Wenn **noteDuration** und **noteFreq** bestimmt sind, kann über den Befehl **FREQOUT** die Note gespielt werden.

```
FREQOUT 9, noteDuration, noteFreq
```

Bei jedem Schleifendurchlauf des Hauptprogramms wird der Wert von **index** um eins erhöht. Startet das Hauptprogramm erneut, wird über die Variable **index** die folgende Note eingelesen.

```
index = index + 1

LOOP
```

Du bist dran – Experimentieren mit Tempo und Melodie

Eine gespielte Notenlänge steht immer in direktem Bezug zum angegebenen Tempo. Das Tempo lässt sich über die Konstante **`WholeNote`** verändern. Wird sie auf 2250 erhöht, nimmt das Tempo ab und das Lied wird langsamer gespielt. Wird sie auf 1750 erniedrigt, nimmt das Tempo zu und das Lied wird schneller abgespielt.

√ Speicher NotesAndDurations.bs2 unter dem Namen ab: NotesAndDurationsYourTurn.bs2.
√ Veränder das Tempo von NotesAndDurationsYourTurn.bs2 durch Anpassung des Wertes von **`WholeNote`**. Wähle: 1500, 1750, 2000 und 2250.
√ Führe das Programm nach jeder Veränderung der Konstanten erneut aus und entscheide, welche Version am besten klingt.

Die Eingabe von Musikdaten gestaltet sich wesentlich einfacher, wenn nur Noten und Notenlängen eingegeben werden müssen. Im folgenden Beispiel werden die ersten acht Noten aus Beethovens 5. Symphony angegeben.


```
Notes     DATA "G","G","G","e","P","F","F","F","D","Q"

Durations DATA   8,  8,  8,  2,  8,  8,  8,  8,  2

WholeNote CON   2000
```

√ Speicher das veränderte Programm Beethoven'sFifth.bs2.
√ Ersetze die Direktiven **`Notes`** und **`Durations DATA`** und die Konstante **`WholeNote`** durch die oben dargestellten Zeilen.
√ Führe das Programm aus. Hört es sich vertraut an?

Notenmerkmale hinzufügen

Das gerade abgeschlossene Beispielprogramm benötigte Noten, Notenlängen und Pausen. Das Tempo wurde über eine ganze Note bestimmt. Handys spielen Klingeltöne ab, die drei zusätzliche Merkmale aufweisen, auf die wir in den bisherigen Beispielen nicht eingegangen sind:

- Sie spielen punktierte Noten.
- Sie bestimmen die Länge einer ganzen Note über einen Wert, der Tempo genannt wird.
- Sie spielen Noten über mehr als nur eine Oktave.

Der Zusatz “punktiert” weist darauf hin, dass die Note um den halben Wert des angegebenen Notenwertes länger gespielt werden muss. Die Tonlänge einer punktierten Viertelnote setzt sich zusammen aus dem Wert einer Viertelnote plus dem Wert einer Achtelnote. Eine punktierte halbe Note hat die Länge einer halben Note plus einer Viertelnote. Über eine entsprechende Datentabelle können alle punktierten und nicht punktierten Noten erfasst werden; im folgenden Beispiel steht die 0 für eine nicht punktierte und die 1 für eine punktierte Note:

```
Dots            DATA    0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,
                        0,  0,  0,  1,  0
```

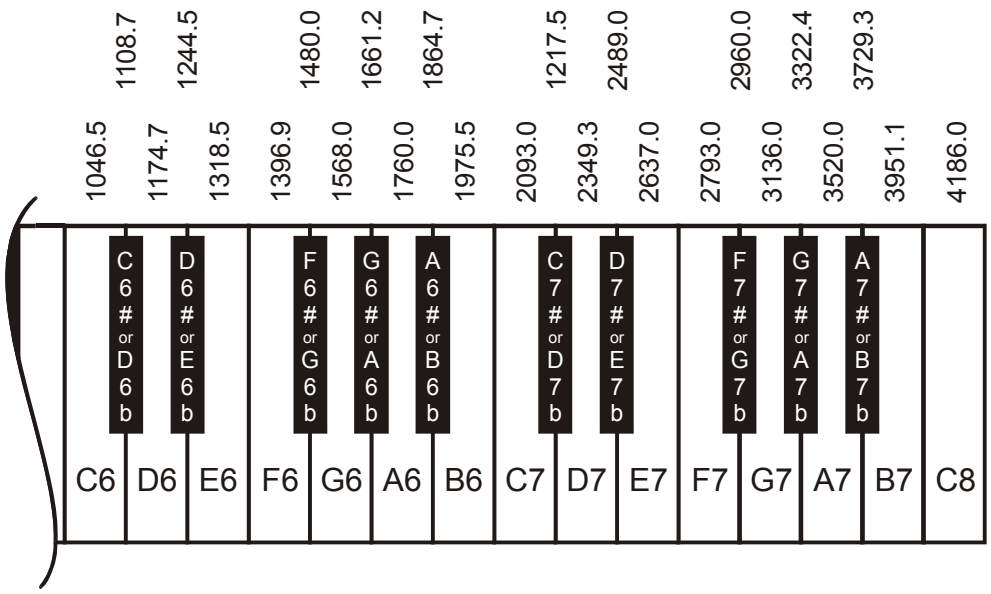
Handys definieren typischerweise das Tempo für einen Song in Beats pro Minute; das ist mit der Aussage vergleichbar: Viertelnoten pro Minute.

```
      BeatsPerMin   CON  200
```

Abbildung 8-4 ist eine Kopie der **Abbildung 8-3** von Seite 245. Es zeigt die 6. und 7. Oktave einer Klaviatur. Diese beiden Oktaven werden von einem Schallwandler am klarsten wiedergegeben. Es folgt ein Beispiel für eine **DATA** Direktive, die im Abschnitt “Du bist dran” benötigt wird, um Töne, die mehr als eine Oktave auseinander liegen, spielen zu können.

```
Octaves         DATA    6,  7,  6,  6,  6,  6,  6,  6,  6,  7,  6,
                        6,  6,  6
```

Abbildung 8-4
Klaviertasten (6. und 7. Oktave) und die zugehörigen Frequenzen



Programmbeispiel: MusicWithMoreFeatures.bs2

Dieses Beispielprogramm spielt die ersten Noten aus dem Lied: *For He's a Jolly Good Fellow*. Alle Noten werden in der 7. Oktave gespielt, einige von ihnen sind gepunktet. Im Abschnitt „Du bist dran" wird in einem Beispiel mit Noten experimentiert, die über mehr als eine Oktave gehen und punktiert sind.

- √ Führe das Programm MusicWithMoreFeatures.bs2 aus.
- √ Zähle die Notenwerte und versuche die punktierten Noten herauszuhören.
- √ Höre dir auch die Noten in der 7. Oktave an. Versuche mindestens eine dieser Noten in die 6. Oktave zu setzen. Die Veränderung bei der Tonwiedergabe wird sehr deutlich ausfallen.

```
' What's a Microcontroller - MusicWithMoreFeatures.bs2
' Play the beginning of For He's a Jolly Good Fellow.

'{$STAMP BS2}
```

```
'{$PBASIC 2.5}

DEBUG "Program Running!"

Notes            DATA    "C","E","E","E","D","E","F","E","E","D","D",
                         "D","C","D","E","C","Q"
Octaves          DATA      7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,
                           7,  7,  7,  7,  7
Durations        DATA      4,  2,  4,  4,  4,  4,  2,  2,  4,  2,  4,
                           4,  4,  4,  2,  2
Dots             DATA      0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,
                           0,  0,  0,  1,  0

BeatsPerMin      CON     320

index            VAR     Byte
offset           VAR     Nib

noteLetter       VAR     Byte
noteFreq         VAR     Word
noteDuration     VAR     Word
noteOctave       VAR     Nib
noteDot          VAR     Bit

wholeNote        VAR     Word
wholeNote = 60000 / BeatsPerMin * 4

DO UNTIL noteLetter = "Q"

  READ Notes + index, noteLetter

  LOOKDOWN noteLetter,  [ "C",  "d",  "D",  "e",  "E",
                          "F",  "g",  "G",  "a",  "A",
                          "b",  "B",  "P",  "Q"     ], offset

  LOOKUP offset,       [ 4186, 4435, 4699, 4978, 5274,
                         5588, 5920, 6272, 6645, 7040,
                         7459, 7902,    0,    0     ], noteFreq

  READ Octaves + index, noteOctave
  noteOctave = 8 - noteOctave
  noteFreq = noteFreq / (DCD noteOctave)

  READ Durations + index, noteDuration
  noteDuration = WholeNote / noteDuration

  READ Dots + index, noteDot
  IF noteDot = 1 THEN noteDuration = noteDuration * 3 / 2

  FREQOUT 9, noteDuration, noteFreq
```

```
  index = index + 1

LOOP

END
```

Arbeitsweise von MusicWithMoreFeatures.bs2

Weiter unten siehst du die musikalischen Daten für das komplette Lied. Zu jeder Note in der Direktive **`Notes DATA`** gibt es einen korrespondierenden Eintrag in den Direktiven **`Octaves, Durations`** und **`Dots DATA`**. Die erste Note ist zum Beispiel ein C in der 7. Oktave; es handelt sich um eine Viertelnote ohne Punktierung. Ein weiteres Beispiel: die zweitletzte Note (ohne das "Q") ist ein E in der 7. Oktave. Es ist eine halbe punktierte Note. Es gibt zusätzlich die Konstante **`BeatsPerMin`**, die das Tempo dieses Liedes festlegt.

```
Notes           DATA    "C","E","E","E","D","E","F","E","E","D","D",
                        "D","C","D","E","C","Q"
Octaves         DATA      7,  7,  7,  7,  7,  7,  7,  7,  7,  7,
                          7,  7,  7,  7,  7
Durations       DATA      4,  2,  4,  4,  4,  4,  2,  2,  4,  2,  4,
                          4,  4,  4,  2,  2
Dots            DATA      0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,
                          0,  0,  0,  1,  0

BeatsPerMin     CON     320
```

Im vorherigen Beispielprogramm benutzten wir die Konstante **`WholeNote`**. Dieses mal benutzen wir sie als Variable, in der die Notenlänge einer ganzen Note in ms abgelegt ist. Nachdem der Wert berechnet worden ist, werden über **`WholeNote`** alle anderen Notenlängen bestimmt, wie schon im vorherigen Programm. Die Variablen **`index`**, **`offset`**, **`noteLetter`** und **`noteDuration`** werden in der gleichen Art wie im vorherigen Programm eingesetzt. Bei der Variablen **`noteFreq`** ist jetzt darauf zu achten, in welcher Oktave die jeweilige Note gespielt werden soll. Die Variablen **`noteOctave`** und **`noteDot`** wurden hinzugefügt, um Oktavierungen und Punktierungen bearbeiten zu können.

```
      wholeNote     VAR  Word

      index         VAR  Byte
      offset        VAR  Nib

      noteLetter    VAR  Byte
      noteFreq      VAR  Word
      noteDuration  VAR  Word
```

```
noteOctave    VAR  Nib
noteDot       VAR  Bit
```

Die Variable **wholeNote** wird mit Hilfe von **BeatsPerMin** berechnet. Das Tempo des Liedes ist in Beats pro Minute angegeben; das Programm dividiert 60000 ms durch **BeatsPerMin** und multipliziert das Ergebnis mit 4. Das Ergebnis gibt den Wert für eine ganze Note an.

```
wholeNote = 60000 / BeatsPerMin * 4
```

Mathematische Terme werden von links nach rechts abgearbeitet. Im Term **wholeNote = 60000 / beatsPerMin * 4**, berechnet die BASIC Stamp zuerst **60000 / beatsPerMin**. Dann wird der Quotient mit 4 multipliziert.

Klammerterme werden zuerst berechnet. Soll zuerst 4 mit beatsPerMin multipliziert werden, geschieht dies über die Klammersetzung: **wholeNote = 60000 / (beatsPerMin * 4)**.

Die folgenden Zeilen werden wie im vorangegangenen Programm abgearbeitet:

```
DO UNTIL noteLetter = "Q"

  READ Notes + index, noteLetter

  LOOKDOWN noteLetter,  [ "C",  "d",  "D",  "e",  "E",
                          "F",  "g",  "G",  "a",  "A",
                          "b",  "B",  "P",  "Q"     ], offset
```

Da die Noten jetzt unterschiedlich oktaviert sind, muss der Teil, in dem die Notenfrequenz bestimmt wird, geändert werden. Die Werte der **LOOKUP** Tabelle enthalten Frequenzen der 8. Oktave. Diese werden durch 1, 2, 4, 8 ... dividiert, wenn die zu spielenden Noten in der 8., 7., 6., 5., ... Oktave liegen. Als nächstes folgt die Division. Der Befehl **LOOKUP** legt zunächst eine Note der 8. Oktave in der Variablen **noteFreq** ab.

```
LOOKUP offset,        [ 4186, 4435, 4699, 4978, 5274,
                        5588, 5920, 6272, 6645, 7040,
                        7459, 7902,    0,    0     ], noteFreq
```

Es folgt die korrekte Oktavierung der Variablen **noteFreq**. Dazu holt sich der Befehl **READ** als erstes den in **Octaves DATA** abgelegten Oktavwert; ein Wert zwischen 5 und 8.

```
READ Octaves + index, noteOctave
```

Abhängig von der Oktave, wird **noteFreq** durch 1, 2, 4 oder 8 dividiert. Das Ziel ist die Division durch $2^0 = 1$, $2^1 = 2$, $2^2 = 4$ oder $2^3 = 8$. In der unten dargestellten Anweisung

wird von der Zahl 8 der Wert von **noteOctave** (ein Wert zwischen 5 und 8) subtrahiert. War **noteOctave** vorher 8, so beträgt sein neuer Wert jetzt 0. War **noteOctave** vorher 7, so ist er neu 1 usw. .

```
noteOctave = 8 - noteOctave
```

Mit dieser Zwischenrechnung erhalten wir **noteOctave** Werte, die als Exponenten von 2 genommen werden können. Wie aber lassen sich unter PBASIC Zahlen potenzieren? Die Antwort lautet: Mit dem **DCD** Operator. **DCD 0** ist 1, **DCD 1** ist 2, **DCD 2** ist 4 und **DCD 3** ist 8. Division von **noteFreq** durch **DCD noteOctave** bedeutet, dass durch 1, 2, 4 oder 8 geteilt und damit **noteFreq** auf den richtigen Frequenzwert, entsprechend der vorgegebenen Oktave, gesetzt wird. Über das Debug Terminal wirst du im Abschnitt "Du bist dran" einen näheren Einblick in die eben besprochene Arbeitsweise bekommen.

```
noteFreq = noteFreq / (DCD noteOctave)
```

Woher soll ich wissen, wie der DCD Operator einzusetzen ist? Durch Lernen und Üben! Jedes mal wenn du einen neuen Befehl, Operator oder ein anderes Schlüsselwort in einem Beispiel kennenlernst, schlage seine Bedeutung im BASIC Stamp Manual nach. Informiere dich und versuche ihn in einem eigenen Programm einzusetzen. Gewöhne es dir an, das BASIC Stamp Manual regelmäßig zu nutzen und die kurzen Beispielprogramme auszuprobieren. Das ist der beste Weg mit den verschiedenen Befehlen und Operatoren und ihren Arbeitsweisen schnell vertraut zu werden. Indem du so verfährst, wirst du deine Programmierwerkzeuge zur Lösung von Problemen ständig erweitern.

Die ersten beiden Programmzeilen, die die Tonlänge bestimmen, gleichen den Programmzeilen des vorherigen Programmbeispiels. Neu ist hier, dass die Noten jetzt auch gepunktet sein können, das heißt, ihre Tonlänge erhöht sich um den Faktor 1,5. Mit dem Befehl **READ** greifen wir auf die mit der **Dots DATA** Direktive im **EEPROM** abgespeicherten Werte zu. Über eine **IF…THEN** Anweisung wird immer dann, wenn der Werte der **noteDot** Variable identisch 1 ist, mit 3 multipliziert und anschließend durch 2 geteilt.

```
READ Durations + index, noteDuration
noteDuration = WholeNote / noteDuration

READ Dots + index, noteDot
IF noteDot = 1 THEN noteDuration = noteDuration * 3 / 2
```

Ganzzahlige Mathematik Die BASIC Stamp kann reelle Zahlen wie 1,5 nicht verarbeiten. Mathematische Operationen lassen sich nur mit ganzen Zahlen …, -5, -4, -3, -2, -1, 0, 1, 2, 3, … durchführen. Eine Multiplikation mit 1,5 (diese Zahl entspricht dem Bruch 3/2) kann auch als Multiplikation mit 3 und Division durch 2 aufgefasst werden.

Es gibt verschiedene Möglichkeiten, Bruchzahlen mit der BASIC Stamp zu verarbeiten. Mit Hilfe ganzer Zahlen kann die BASIC Stamp so programmiert werden, dass der rationale Anteil einer Zahl ermittelt wird. Informationen hierzu finden sich im Student Guide *Basic Analog and Digital*. Mit den beiden Operatoren ****** and ***/** lassen sich rationale Zahlen leichter bearbeiten. Nähere Informationen hierzu findest du im Student Guide *Applied Sensors* und im BASIC Stamp Manual.

Des Rest dieses Beispielprogramms ist identisch mit dem vorherigen Programmbeispiel:

```
    FREQOUT 9, noteDuration, noteFreq
    index = index + 1
  LOOP
  END
```

Du bist dran – Eine Melodie über mehr als eine Oktave spielen

Im Musikprogramm WithMoreFeatures.bs2 wurden Pausennoten benutzt, aber der Tonumfang beschränkte sich auf eine Oktave. Die meisten Noten der folgenden Melodie "Take Me Out to the Ball Game" werden in der 6. Oktave gespielt. Zwei Noten werden in der 7. Oktave gespielt und verändern den Song deutlich.

√ Speicher das Beispielprogramm unter dem folgenden Namen ab: MusicWithMoreFeaturesYourTurn.bs2.

√ Tausche die vier Data Direktiven und die Konstante Deklaration gegen die folgenden aus:

```
Notes           DATA    "C","C","A","G","E","G","D","P","C","C","A",
                        "G","E","G","Q"
Octaves         DATA      6,  7,  6,  6,  6,  6,  6,  6,  6,  7,  6,
                          6,  6,  6
Durations       DATA      2,  4,  4,  4,  4,  2,  2,  4,  2,  4,  4,
                          4,  4,  2
Dots            DATA      0,  0,  0,  0,  0,  1,  0,  0,  0,  0,  0,
                          0,  0,  1

BeatsPerMin     CON     240
```

√ Führe das Programm aus und überprüfe die Melodie.

Die beiden Noten in der 7. Oktave geben dem Lied den besonderen Klang. Höre dir an, wie sich der Gesamteindruck des Liedes verändert, wenn die beiden Noten in der 6. Oktave angespielt werden.

- √ Setze die beiden 7-er Werte in der Direktive **`Octaves DATA`** auf 6. Bedenke, dass das Stück "Take Me out to the Ball Game" dadurch seltsam klingen wird.
- √ Führe das Programm aus und höre dir das Lied genau an.
- √ Tausche die in **`Octaves DATA`** geänderten Werte wieder mit den Originalwerten.
- √ Führe das Programm erneut aus und überzeuge dich, dass das Lied richtig klingt.

ÜBUNG 5: HANDY KLINGELTÖNE PROGRAMMIEREN

Viele Handys spielen Musiken ab, die von Webseiten heruntergeladen werden können. Der Computer überträgt die Daten der Noten an den Mikrocontroller im Handy und spielt sie ab, sobald ein Anruf oder eine SMS eingeht. Man bezeichnet diese Töne als Klingeltöne.

Eine sehr weit verbreitete Methode Melodien zu komponieren, aufzunehmen und zu verbreiten arbeitet mit Textzeilen, die jede Note eines Liedes beschreiben. Es folgt ein Beispiel für die ersten Noten aus Beethovens 5. Symphonie im RTTTL Format:

```
Beethoven5:d=8,o=7,b=125:g,g,g,2d#,p,f,f,f,2d
```

Das Speicherformat für die Musikdaten heißt RTTTL, ein Akronym für *Ringing Tone Text Transfer Language*; seine Spezifikation ist im World Wide Web veröffentlicht. RTTTL Dateien werden über das World Wide Web vertrieben und lassen sich häufig kostenlos herunterladen. Dort finden sich auch Freewareprogramme, mit denen sich diese Dateien zusammensetzen bzw. emulieren und auf das Handy übertragen lassen. Im Anhang G wird noch einmal zusammenfassend beschrieben, wie Noten, Notenlängen, Pausen, Tempi und punktierte Noten in einer RTTTL Datei abgespeichert werden.

In dieser Übung lernst du einige PBASIC Programmiertechniken kennen, mit denen sich unterschiedliche Textstellen in einem Text herausfinden lassen. Die Fähigkeit, verschiedene Zeichen oder ganze Gruppen von Zeichen innerhalb eines Textes zu erkennen und, darauf basierend, Handlungen auszuführen, lässt sich zukünftig vielfach verwerten. Für unsere Übung ist es *die* Schlüsselfunktion, mit der RTTTL-formatierte Klingeltöne (wie Beethovens fünfte) in Musik konvertiert werden. Am Ende dieser

Übung befindet sich ein Anwendungsprogramm, mit dem sich RTTTL formatierte Klingeltöne abspielen lassen.

Die Select ... CASE Anweisung

Mit Hilfe der Anweisung **SELECT…CASE** lassen sich wohl am einfachsten Zeichen oder Werte erkennen. Damit besitzen wir ein Werkzeug, mit dem sich RTTTL Klingeltöne in Musiknoten konvertieren lassen. Eine **SELECT…CASE** Anweisung wird eingesetzt:

- Um eine Variable oder einen Ausdruck auszuwählen.
- Um eine Variable oder einen Ausdruck auf der Basis einer Bedingung zu berechnen.
- Um einen bestimmten Programmteile auszuführen, je nachdem, welche Bedingung durch den Variablenwert erfüllt wurde.

Syntax von **SELECT…CASE**:

```
SELECT Ausdruck
  CASE Bedingung(en)
    Anweisung(en)
ENDSELECT
```

Die Arbeitsweise der **SELECT…CASE** Anweisung lässt sich an den beiden folgenden Beispielprogrammen studieren. SelectCaseWithValues.bs2 erfragt einen nummerischen Wert über das Debug Terminal und gibt den dafür notwendigen Variablentyp zurück. SelectCaseWithCharacters.bs2 meldet zurück, ob es sich bei dem über das Debug Terminal eingegebene Zeichen um einen großen oder kleinen Buchstaben, eine Zahl oder ein Interpunktionszeichen handelt.

Benutze für den Dialog mit der BASIC Stamp den oberen Teil des Fensters im Debug Terminal. Die Darstellungsfenster für Senden und Empfang zeigt Abbildung 8-5.

Darstellungsfenster

Senden →

Empfangen →

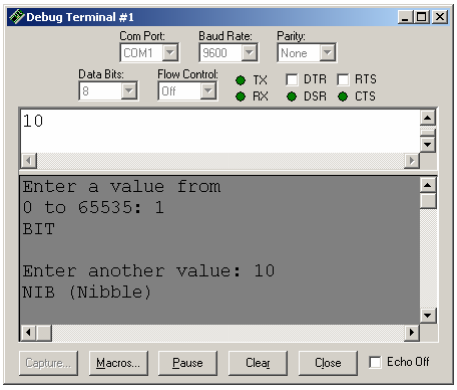


Abbildung 8-5
Mitteilungen an die BASIC Stamp übertragen

Klicke in das Fenster Senden (oben) und gib dort einen Wert oder ein Zeichen für die BASIC Stamp ein.

Programmbeispiel: SelectCaseWithValues.bs2

√ Führe das Programm SelectCaseWithValues.bs2 aus.
√ Klicke im Debug Terminal auf die Windowsoberfläche Senden.
√ Gib einen Wert zwischen 0 und 65535 ein und bestätige mit der Eingabetaste.

Was passiert, wenn der eingegebene Zahlenwert größer als 65535 ist? Wird 65536 eingegeben, speichert die BASIC Stamp den Wert 0. Bei Eingabe von 65537 speichert die BASIC Stamp die Zahl 1 und so weiter. Ist eine eingegebene Zahl zu groß für den angegebenen Variablentyp sprechen wir von einem Überlauf .

√ Nachdem du deine Zahleingabe im Debug Terminal gemacht hast, trifft das Beispielprogramm eine Entscheidung und gibt den zugrundliegenden Variablentyp im Display aus. Benutze zur Überprüfung auf Richtigkeit die Table 2-2.

Table 2-2: Variablentypen und deren Wertebereiche

Variablentyp	Wertebereich
Bit	0 to 1
Nib	0 to 15
Byte	0 to 255
Word	0 to 65535

```
' What's a Microcontroller - SelectCaseWithValues.bs2
```

```
' Enter a value and see the minimum variable size required to hold it.

'{$STAMP BS2}
'{$PBASIC 2.5}

value          VAR     Word

DEBUG "Enter a value from", CR,
      "0 to 65535: "

DO

  DEBUGIN DEC value

  SELECT value

    CASE 0, 1
      DEBUG "Bit", CR
      PAUSE 100

    CASE 2 TO 15
      DEBUG "Nib (Nibble)", CR
      PAUSE 200

    CASE 16 TO 255
      DEBUG "Byte", CR
      PAUSE 300

    CASE 256 TO 65535
      DEBUG "Word", CR
      PAUSE 400

  ENDSELECT

  DEBUG CR, "Enter another value: "

LOOP
```

Arbeitsweise von SelectCaseWithValues.bs2

Value wird als Variable vom Typ Word deklariert.

```
value VAR Word
```

Der Befehl **DEBUGIN** übernimmt die eingegebene Zahl und legt sie in der Variablen **value** ab.

```
DEBUGIN DEC value
```

Die Anweisung **SELECT** liest den Wert der Zahl, die in der Variablen **value** abgelegt wurde und vergleicht ihn mit den CASE Anweisungen.

```
SELECT value
```

Die erste CASE Anweisung überprüft, ob die Variable **value** 0 oder 1 ist. Ist dies der Fall, werden die Befehle **DEBUG** und **PAUSE** ausgeführt. In jedem anderen Fall wird zur nächsten CASE Anweisung gegangen.

```
CASE 0, 1
  DEBUG "BIT", CR
  PAUSE 100
```

Die zweite CASE Anweisung überprüft, ob die Variable **value** zwischen 2 und 15 liegt. Ist dies der Fall, werden die Befehle **DEBUG** und **PAUSE** ausgeführt..

```
CASE 2 to 15
  DEBUG "NIB (Nibble)", CR
  PAUSE 200
```

Sind alle CASE Anweisungen abgearbeitet, beendet das Schlüsselwort **ENDSELECT** die **SELECT..CASE** Anweisung.

```
ENDSELECT
```

Programmbeispiel: SelectCaseWithCharacters.bs2

In diesem Beispielprogramm wird jedes Zeichen, das über die Windowsoberfläche Senden des Debug Terminals eingegeben wird, näher bestimmt. Groß- und Kleinbuchstaben, ganze Zahlen und einige Interpunktionszeichen werden erkannt. Wird ein Zeichen vom Programm nicht identifiziert, fordert das Programm zu einer Neueingabe auf (Eingabe eines anderen Zeichens).

- √ Gib das Programm SelectCaseWithCharacters.bs2 ein und führe es aus.
- √ Klicke in die Windowsoberfläche Senden des Debug Terminals, gib ein beliebiges Zeichen ein und beobachte die Reaktion.

```
' What's a Microcontroller - SelectCaseWithCharacters.bs2
' Program that can identify some characters: case, digit, punctuation.

'{$STAMP BS2}
'{$PBASIC 2.5}

character        VAR     Byte
```

```
DEBUG "Enter a character: ", CR

DO

  DEBUGIN character

  SELECT character

    CASE "A" TO "Z"
      DEBUG CR, "Upper case", CR

    CASE "a" TO "z"
      DEBUG CR, "Lower case", CR

    CASE "0" TO "9"
      DEBUG CR, "Digit", CR

    CASE "!", "?", ".", ","
      DEBUG CR, "Punctuation", CR

    CASE ELSE
      DEBUG CR, "Character not known.", CR,
                "Try a different one."

  ENDSELECT

  DEBUG CR, "Enter another character", CR

LOOP
```

Arbeitsweise von SelectCaseWithCharacters.bs2

Vergleicht man dieses Programm mit SelectCaseWithValues.bs2, treten ein paar Unterschiede auf. Der Name der Variablen **`value`** wurde in **`character`** geändert, der Variablentyp von Word in Byte geändert. Dies geschah, da alle Zeichen in PBASIC vom Typ Byte sind. Die **`SELECT`** Anweisung führt die CASE-Überprüfung über die Variable **`character`** durch.

```
SELECT character
```

Die Anführungszeichen teilen dem BASIC Stamp Editor mit, dass es sich um Buchstaben handelt.

```
SELECT character

  CASE "A" to "Z"
    DEBUG CR, "Upper case", CR

  CASE "a" to "z"
```

```
    DEBUG CR, "Lower case", CR

  CASE "0" to "9"
    DEBUG CR, "Digit", CR

  CASE "!", "?", ".", ","
    DEBUG CR, "Punctuation", CR
```

Es gibt eine **CASE** Anweisung, die im vorherigen Beispiel nicht vorhanden war:

```
  CASE ELSE
    DEBUG CR, "Character not known.", CR,
              "Try a different one."
```

In dieser **CASE** Anweisung wird dem **SELECT** Anweisungsteil mitgeteilt, was getan werden soll, wenn keiner der o.g. Fälle (CASES) eingetreten ist. Die Eingabe von % oder $ würde automatisch auf diesen Anweisungsteil zugreifen.

Du bist dran – Auswahl nach Bereichen

√ Änder die Anweisung **SELECT…CASE** im Programm SelectCaseWithCharacters.bs2 so ab, dass "Sonderzeichen" ausgegeben wird, wenn eines der folgenden Zeichen eingegeben wurde: @, #, $, %, '^' , &, *, (,), _, or +.

Abspielprogramm für RTTTL Klingeltöne

Weiter unten ist eine RTTTL Datei mit den Musikinformationen, die im folgenden Beispielprogramm benutzt werden, angegeben. Weitere fünf **RTTTL_File DATA** Direktiven, die über den Abschnitt "Du bist dran" ausprobiert werden können, befinden sich am Ende dieses Abschnittes. Das vorliegende Musikstück „Reveille" mag dir aus Filmen oder dem Fernsehen bekannt vorkommen.

```
RTTTL_File      DATA    "Reveille:d=4,o=7,b=140:8g6,8c,16e,16c,8g6,8e,",
                        "8c,16e,16c,8g6,8e,8c,16e,16c,8a6,8c,e,8c,8g6,",
                        "8c,16e,16c,8g6,8e,8c,16e,16c,8g6,8e,8c,16e,",
                        "16c,8g6,8e,c,p,8e,8e,8e,8e,g,8e,8c,8e,8c,8e,8c,",
                        "e,8c,8e,8e,8e,8e,8e,g,8e,8c,8e,8c,8g6,8g6,c."
```

Programmbeispiel: MicroMusicWithRtttl.bs2

Dieses Anwendungsprogramm ist ziemlich lang; es wäre deshalb angebracht, die neueste Version direkt von der folgenden Website herunterzuladen: www.parallax.com → Downloads → Educational Curriculum. Orientiere dich am Link *Selected Example Programs* in der Nähe des PDF Downloadbereichs *What's a Microcontroller*. Du sparst

viel Zeit, wenn du dir das Programm herunterlädst und mit dem BASIC Stamp Editor öffnest. Alternativ kannst du das über vier Seiten gehende Programm natürlich auch mit der Hand eingeben und anschließend nach Fehlern durchsuchen.

- √ Öffne mit dem BASIC Stamp Editor das aus dem Internet heruntergeladene Programm MicroMusicWithRtttl.bs2 oder gib das Programm nach der unten abgedruckten Vorlage sorgfältig mit der Hand ein.
- √ Führe das Programm aus und identifiziere es als das Stück “Reveille bugle call“.
- √ Im Abschnitt “Du bist dran” findest du weitere Melodien; probiere auch sie aus (RTTTL_File DATA directives).

```
' What's a Microcontroller - MicroMusicWithRtttl.bs2
' Play Nokia RTTTL format ringtones using DATA.

'{$STAMP BS2}
'{$PBASIC 2.5}
DEBUG "Program Running!"

' -----[ I/O Definitions ]--------------------------------------------------

  SpeakerPin     CON     9                  ' Piezospeaker connected to P9.

' -----[ Variables ]-------------------------------------------------------

  counter        VAR     Word               ' General purpose counter.
  char           VAR     Byte               ' Variable stores characters.
  index          VAR     Word               ' Index for pointing at data.

  noteLetter     VAR     Byte               ' Stores note character.
  noteFreq       VAR     Word               ' Stores note frequency.
  noteOctave     VAR     Word               ' Stores note octave.

  duration       VAR     Word               ' Stores note duration.
  tempo          VAR     Word               ' Stores tempo.

  default_d      VAR     Byte               ' Stores default duration.
  default_o      VAR     Byte               ' Stores default octave.
  default_b      VAR     Word               ' Stores default beats/min.

' -----[ EEPROM Data ]-----------------------------------------------------

  RTTTL_File     DATA    "Reveille:d=4,o=7,b=140:8g6,8c,16e,16c,8g6,8e,",
                         "8c,16e,16c,8g6,8e,8c,16e,16c,8a6,8c,e,8c,8g6,",
                         "8c,16e,16c,8g6,8e,8c,16e,16c,8g6,8e,8c,16e,",
                         "16c,8g6,8e,c,p,8e,8e,8e,8e,g,8e,8c,8e,8c,8e,8c,",
                         "e,8c,8e,8e,8e,8e,8e,g,8e,8c,8e,8c,8g6,8g6,c."
```

```
  Done            DATA    ",q,"

  Notes           DATA     "p",       "a",       "#",       "b",
                           "c",       "#",       "d",       "#",
                           "e",       "f",       "#",       "g",
                           "#"
  Octave8         DATA    Word 0,    Word 3520, Word 3729, Word 3951,
                          Word 4186, Word 4435, Word 4699, Word 4978,
                          Word 5274, Word 5588, Word 5920, Word 6272,
                          Word 6645

' -----[ Initialization ]---------------------------------------------------

  counter = 0                                ' Initialize counter.

  GOSUB FindEquals                           ' Find first '=' in file.
  GOSUB ProcessDuration                      ' Get default duration.
  GOSUB FindEquals                           ' Find next '='.
  GOSUB ProcessOctave                        ' Get default octave.
  GOSUB FindEquals                           ' Find last '='.
  GOSUB GetTempo                             ' Get default tempo.

' -----[ Program Code ]-------------------------------------------------------

  DO UNTIL char = "q"                        ' Loop until 'q' in DATA.
    GOSUB ProcessDuration                    ' Get note duration.
    GOSUB ProcessNote                        ' Get index value of note.
    GOSUB CheckForDot                        ' If dot, 3/2 duration.
    GOSUB ProcessOctave                      ' Get octave.
    GOSUB PlayNote                           ' Get freq, play note, next.
  LOOP                                       ' End of main loop.

  END                                        ' End of program.

' -----[ Subroutine - Find Equals Character ]----------------------------

  FindEquals:                                ' Go through characters in
                                             ' RTTTL file looking for
    DO                                       ' '='.  Increment counter
      READ RTTTL_File + counter, char        ' until '=' is found, then
      counter = counter + 1                  ' return.
    LOOP UNTIL char = "="

    RETURN

' -----[ Subroutine - Read Tempo from RTTTL Header ]----------------------
' Each keyboard character has a unique number called an ASCII value.
' The characters 0, 1, 2,...9 have ASCII values of 48, 49, 50,...57.
' You can always convert from the character representing a digit to
' to its value by subtracting 48 from the variable storing the digit.
```

```
' You can examine this by comparing DEBUG DEC 49 and DEBUG 49.

  GetTempo:                                 ' Parse RTTTL file for Tempo.
                                            ' Convert characters to
    default_b = 0                           ' digits by subtracting 48
    DO                                      ' from each character's ASCII
      READ RTTTL_File + counter, char       ' value.  Iteratively multiply
      IF char = ":" THEN                    ' each digit by 10 if there
        default_b = default_b / 10          ' is another digit, then add
        counter = counter + 1               ' the most recent digit to
        EXIT                                ' one's column.
      ENDIF                                 ' For example, the string
        default_b = default_b + char - 48   ' "120" is (1 X 10 X 10)
        counter = counter + 1               ' + (2 X 10) + 0.  The '1'
        default_b = default_b * 10          ' is converted first, then
    LOOP UNTIL char = ":"                   ' multiplied by 10.  The '2'
                                            ' is then converted/added.
    RETURN                                  ' 0 is converted/added, done.

' -----[ Subroutine - Look up Octave ]-----------------------------------

  ProcessOctave:                            ' Octave may or may not be
                                            ' included in a given note
    READ RTTTL_File + counter, char         ' because any note that is
    SELECT char                             ' played in the default
      CASE "5" TO "8"                       ' octave does not specify
        noteOctave = char - "0"             ' the octave.  If a char
        counter = counter + 1               ' from '5' to '8' then use
      CASE ELSE                             ' it, else use default_o.
        noteOctave = default_o              ' Characters are converted
    ENDSELECT                               ' to digits by subtracting
    IF default_o = 0 THEN                   ' '0', which is the same as
      default_o = noteOctave                ' subtracting 48. The first
    ENDIF                                   ' time this subroutine is
                                            ' called, default_o is 0.
    RETURN                                  ' If 0, then set default_o.

' -----[ Subroutine - Find Index of Note ]-------------------------------

  ProcessNote:                              ' Set index value for lookup
                                            ' of note frequency based on
    READ RTTTL_File + counter, char         ' note character. If 'p',
    SELECT char                             ' index is 0.  If 'a' to 'g',
      CASE "p"                              ' read character values in
        index = 0                           ' DATA table and find match.
        counter = counter + 1               ' Record index value when
      CASE "a" TO "g"                       ' match is found.  If next
        FOR index = 1 TO 12                 ' char is a sharp (#), add
          READ Notes + index, noteLetter    ' 1 to the index value to
          IF noteLetter = char THEN EXIT    ' increase the index (and
        NEXT                                ' frequency) by 1 notch.
```

```
        counter = counter + 1                ' As with other subroutines,
        READ RTTTL_File + counter, char      ' increment counter for each
        SELECT char                          ' character that is processed.
          CASE "#"
            index = index + 1
            counter = counter + 1
        ENDSELECT
    ENDSELECT

    RETURN

' -----[ Subroutine - Determine Note Duration ]--------------------------

  ProcessDuration:                           ' Check to see if characters
                                             ' form 1, 2, 4, 8, 16 or 32.
    READ RTTTL_File + counter, char          ' If yes, then convert from
                                             ' ASCII character to a value
    SELECT char                              ' by subtracting 48. In the
      CASE "1", "2", "3", "4", "8"           ' case of 16 or 32, multiply
        duration = char - 48                 ' by 10 and add the next
        counter = counter + 1                ' digit to the ones column.
        READ RTTTL_File + counter, char
        SELECT char
          CASE "6", "2"
            duration = duration * 10 + char - 48
            counter = counter + 1
        ENDSELECT
        CASE ELSE                            ' If no duration, use
        duration = default_d                 ' use default.
    ENDSELECT

    IF default_d <> 0 THEN                   ' If default_d not defined
      duration = 60000/default_b/duration*3  ' (if default_d = 0), then
    ELSE                                     ' set default_d = to the
      default_d = duration                   ' duration from the d=#.
    ENDIF

    RETURN

' -----[ Subroutine - Check For '.' Indicating 1.5 Duration ]-------------

  CheckForDot:                               ' Check for dot indicating
                                             ' multiply duration by 3/2.
    READ RTTTL_File + counter, char          ' If dot found, multiply by
    SELECT char                              ' 3/2 and increment counter,
      CASE "."                               ' else, do nothing and
        duration = duration * 3 / 2          ' return.
        counter = counter + 1
    ENDSELECT

    RETURN
```

```
' -----[ Subroutine - Find Comma and Play Note/Duration ]-----------------

  PlayNote:                                    ' Find last comma in the
                                               ' current note entry.  Then,
    READ RTTTL_File + counter, char            ' fetch the note frequency
    SELECT char                                ' from data, and play it, or
      CASE ","                                 ' pause if frequency = 0.
        counter = counter + 1
        READ Octave8 + (index * 2), Word noteFreq
        noteOctave = 8 - noteOctave
        noteFreq = noteFreq / (DCD noteOctave)
        IF noteFreq = 0 THEN
           PAUSE duration
        ELSE
          FREQOUT SpeakerPin, duration, noteFreq
        ENDIF
    ENDSELECT

    RETURN
```

Arbeitsweise von MicroMusicWithRtttl.bs2

Der Spaßfaktor dieses Programms ist hoch, und es enthält Programmiercode, den du – bei ein wenig Praxis - bald auch selbst schreiben kannst.

Wenn du dir die Programmzeilen genau anschaust, wirst du feststellen, dass alle vorkommenden Befehle und Operatoren bekannt sind. Die folgende Liste zeigt noch einmal alle Befehle und Anweisungen des Anwendungsprogramms:

- Kommentare, um das Programm verständlich zu machen
- Konstanten- und Variablendeklarationen
- **`DATA`** Deklarationen
- **`READ`** Befehle
- **`IF…ELSE…ENDIF`** Schleifen
- **`DO…LOOP`** Schleifen sowohl mit als auch ohne **`WHILE`** und **`UNTIL`**
- Unterprogramme mit **`GOSUB`**, Einsprungmarken und **`RETURN`**
- **`FOR…NEXT`** Schleifen
- **`LOOKUP`** und **`LOOKDOWN`** Befehle
- **`FREQOUT`** und **`PAUSE`** Befehle
- **`SELECT…CASE`**

Du bist dran – Verschiedene Melodien

√ Tausche die **`RTTTL_File DATA`** Direktive in MicroMusicWithRTTTL.bs2 nacheinander durch eine der fünf anderen Melodien aus.

Jeweils nur eine `RTTTL_File DATA` Direktive zur Zeit! Achte darauf die neue **`RTTTL_File DATA`** Direktive gegen die alte auszutauschen und nicht zusätzlich hinzuzufügen.

√ Führe MicroMusicWithRTTTL.bs2 aus und teste jede RTTTL-Datei.

```
RTTTL_File    DATA    "TwinkleTwinkle:d=4,o=7,b=120:c,c,g,g,a,a,2g,f,",
                      "f,e,e,d,d,2c,g,g,f,f,e,e,2d,g,g,f,f,e,e,2d,c,c,",
                      "g,g,a,a,2g,f,f,e,e,d,d,1c"

RTTTL_File    DATA    "FrereJacques:d=4,o=7,b=125:c,d,e,c,c,d,e,c,e,f",
                      ",2g,e,f,2g,8g,8a,8g,8f,e,c,8g,8a,8g,8f,e,c,c,g6",
                      ",2c,c,g6,2c"

RTTTL_File    DATA    "Beethoven5:d=8,o=7,b=125:g,g,g,2d#,p,f,f,f,2d"

RTTTL_File    DATA    "ForHe'sAJollyGoodFellow:d=4,o=7,b=320:c,2e,e,e,",
                      "d,e,2f.,2e,e,2d,d,d,c,d,2e.,2c,d,2e,e,e,d,e,2f,",
                      "g,2a,a,g,g,g,2f,d,2c"

RTTTL_File    DATA    "TakeMeOutToTheBallgame:d=4,o=7,b=225:2c6,c,a6,",
                      "g6,e6,2g.6,2d6,p,2c6,c,a6,g6,e6,2g.6,g6,p,p,a6",
                      ",g#6,a6,e6,f6,g6,a6,p,f6,2d6,p,2a6,a6,a6,b6,c,",
                      "d,b6,a6,g6"
```

RTTTL Dateien herunterladen: Es gibt eine Vielzahl von RTTTL Dateien, die kostenlos aus dem Internet heruntergeladen werden dürfen. Diese Dateien werden von Klingelton-Enthusiasten entwickelt, von denen viele keine Musikexperten sind. Deshalb mag es nicht verwunderlich sein, wenn einige Klingeltöne recht gut klingen, andere aber nicht. Möchtest du eine heruntergeladene RTTTL Datei abspielen, müssen zuerst alle Leerstellen zwischen den Buchstaben gelöscht und anschließend die Textdatei in Anführungszeichen gesetzt werden.

ZUSAMMENFASSUNG

In diesem Kapitel lernten wir Techniken kennen, mit denen sich über eine BASIC Stamp und einen Piezoschallwandler Melodien und Töne erzeugen lassen. Mit dem Befehl **FREQOUT** werden HIGH/LOW Signale an einen Schallwandler geschickt und Soundeffekte und/oder Melodien erzeugt. Über die Argumente des **FREQOUT** Befehls werden die I/O Pins, an die das Signal geschickt wird, die ***Duration*** (Tonlänge) und die Frequenz (***Freq1***) des Tones gesteuert. Das optionale Argument ***Freq2*** wird bei Mischtönen angesprochen.

Soundeffekte lassen sich durch Verstellen von Frequenzen, Tonlängen und Pausen erzeugen. Weitere Effekte erzielt man dadurch, dass eine Frequenz über einen bestimmten Bereich verschoben oder mit einer anderen gemischt wird.

Auch das Aufstellen von Musiknoten hängt von der Frequenz, der Notenlänge und den dazwischen liegenden Pausen ab. Der Wert des Arguments ***Duration*** des Befehls **FREQOUT** wird durch das Tempo des Liedes und die Tonlänge der Note (ganze, halbe, viertel usw.) festgelegt. Der Wert ***Freq1*** einer Note wird bestimmt durch die Bezeichnung der Note und die Oktave. Pausen zwischen den Noten werden über die Länge des Befehls **PAUSE** gesetzt.

Mit eine Folge von **FREQOUT** Befehlen lassen sich einfache Songs über eine BASIC Stamp abspielen. Um Musikdaten zu speichern und wieder abzurufen, gibt es aber bessere Möglichkeiten. Wir haben dazu die Direktive **DATA** zusammen mit den optionalen ***Symbol*** Labeln eingeführt und Werte vom Typ Byte und Word abgespeichert. Mit dem **READ** Befehl wurden Werte zurückgelesen, die vorher über die **DATA** Direktive abgespeichert wurden. Das Argument ***Address*** im Befehl **READ** benutzt immer das optionale ***Symbol*** Label der **DATA** Direktive und kann damit die unterschiedlichen Datentypen auseinander halten. Einige der benutzten Symbollabel hießen **Notes**, **Durations**, **Dots** und **Octaves**.

Musikdaten lassen sich in Formaten abspeichern, die sich an Übersetzungen der Originaldaten anlehnen. Das Notenmaterial wird über die Befehle **LOOKUP** und **LOOKDOWN** in Frequenzen umgesetzt. Mit mathematischen Operationen lassen sich die Werte von Variablen verändern. Teilt man zum Beispiel die Frequenz einer Note durch eine Potenz von zwei, ändert sich die Tonhöhe (Oktavierung). Mathematische Operationen sind auch

bei der Angabe von Notenlängen nützlich, entweder durch eine Tempoangabe oder durch die Angabe eines Tonlängenäquivalents für eine ganze Note.

SELECT…CASE haben wir als ein Entscheidungsverfahren kennengelernt, bei dem über den Wert einer Variablen ein bestimmter Programmblock abgearbeitet wurde. Über ein Programm, das Zeichenketten konvertiert, die musikalische Töne für Handys beschreiben (sog. RTTTL Dateien), lernten wir ein umfangreicheres Programm kennen, in das alle Programmiertechniken dieses Kapitels eingebaut wurden.

Verständnisfragen

1. Wodurch wird ein Ton als hoher/tiefer Ton wahrgenommen?
2. Erkläre im Befehl `FREQOUT 15, 1000, 3000` die Bedeutung jeder Zahl.
3. Wie lässt sich der Befehl **FREQOUT** aus Frage 2 so abändern, dass gleichzeitig zwei Frequenzen ausgegeben werden?
4. Welcher Frequenz entspricht der Ton beim Anschlagen der Taste B6 auf einem Klavier?
5. Wie änderst du die Direktive **DATA** oder den Befehl **READ** ab, wenn Werte vom Typ Word abgespeichert bzw. zurückgeladen werden sollen?
6. Darf in einem Programm mehr als eine **DATA** Direktive angegeben werden? Wenn du mit ja antwortest, wie muss dann der **READ** Befehl konfiguriert werden, um Daten aus einer bestimmten **DATA** Direktive anzusprechen?
7. Was versteht man unter einer Oktave? Wenn du die Frequenz einer bestimmten Note kennst, wie muss die Frequenz dann geändert werden, damit die Note eine Oktave höher klingt?
8. Wie arbeitet der Entscheidungsbefehl **SELECT…CASE**?

Übungen

1. Änder den “Alarm…” Ton aus ActionTones.bs2 so ab, dass die Frequenz nach jedem Programmdurchlauf um 500 Hz angehoben wird.
2. Erkläre, wie man MusicWithMoreFeatures.bs2 verändern muss, damit eine Warnmeldung bei jeder gepunkteten Note, die abgespielt wird, im Debug Terminal erscheint.

Projekt

1. Baue einen tastengesteuerten Tongenerator. Wird ein Taster gedrückt, sollte der Lautsprecher einen 2 kHz Ton für 1/5 Sekunde ausgeben. Wird ein zweiter Taster gedrückt, sollte der Lautsprecher einen 3 kHz Ton für 1/10 Sekunden

ausgeben.

Lösungen

Q1. Druckänderungen in der Luft nimmt unser Ohr als Ton wahr. Ein hoher Ton wird durch schnelle Druckänderungen in der Luft erzeugt, ein niedriger durch langsame Luftdruckänderungen.

Q2. **`FREQOUT 15, 1000, 3000`** sendet ein 3000 Hz Signal an Pin 15 für eine Sekunde (1000 ms).
Die Wirkung jeder Zahl:
15 – Pinnummer 15.
1000 – Tonlänge entspricht 1000 ms oder 1 Sekunde.
3000 – Tonfrequenz in Hertz; hier: 3000 Hz

Q3. Benutze das Argument ***`Freq2`***. Um 3000 Hz und z.B. 2000 Hz zu mischen, fügen wir die zweite Frequenz rechts von der ersten zum Befehl hinzu:
`FREQOUT 15, 1000, 3000, 2000`

Q4. 1975.5 Hz, siehe Abbildung 8-3 auf Seite 245.

Q5. Füge den optionalen Modifier **`Word`** vor jeden Dateneintrag ein.

Q6. Ja. Jede Direktive **`DATA`** muss einen anderen Symbolparameter enthalten. Um festzulegen, von welcher **`DATA`** Direktive die Daten gelesen werden, füge den Symbolparameter dem **`READ`** Befehl hinzu. Beispiel: **`READ Notes, noteLetter.`** In diesem Beispiel ist **`Notes`** der Symbolparameter.

Q7. Eine Oktave besteht aus 8 Ganzton- und 4 Halbtonschritten. Um eine Note um eine Oktave anzuheben, muss die Frequenz mit zwei multipliziert werden.

Q8. **`SELECT...CASE`** wählt eine Variable oder einen Ausdruck, trifft auf der Basis vorgegebener Fälle eine individuelle Entscheidung und führt in Abhängigkeit der Entscheidung unterschiedliche Befehlssequenzen aus.

E1. Dieses Problem lässt sich entweder dadurch lösen, dass man jeden Ton um 500 Hz erhöht oder über eine **`FOR...NEXT`** Schleife mit einem **`STEP`** Wert von 500.

Anpassung **`FOR...NEXT`** Schleife:

```
DEBUG "Increasing alarm...", CR
   PAUSE 100
   FOR frequency = 1500 TO 3000 STEP 500
     FREQOUT 9, 500, frequency
     PAUSE 500
   NEXT
```

Tonerhöhung manuell:

```
DEBUG "Increasing Alarm...",CR
   PAUSE 100
   FREQOUT 9, 500, 1500
   PAUSE 500
   FREQOUT 9, 500, 2000
   PAUSE 500
   FREQOUT 9, 500, 2500
   PAUSE 500
```

```
                                        FREQOUT 9, 500, 3000
                                        PAUSE 500
```

E2. Änder die Zeilen ab, in denen auf gepunktete Noten überprüft wird:

```
READ Dots + index, noteDot
IF noteDot = 1 THEN noteDuration = noteDuration * 3 / 2
```

Add a **DEBUG** command to the **IF...THEN**. Don't forget the **ENDIF**.

```
READ Dots + index, noteDot
IF noteDot = 1 THEN
  noteDuration = noteDuration * 3 / 2
  DEBUG "Dotted Note!", CR
ENDIF
```

P1. Benutze die Schaltung Piezoschallwandler aus Abbildung 8-2, S. 237; Tasterschaltung aus Abbildung 4-20, S. 136.

```
' What's a Microcontroller - Ch8Prj01_PushButtonToneGenerator.bs2
' P4 Pressed:  2 kHz beep for 1/5 second. 2 kHz = 2000 Hz.
'              1/5 s = 1000 / 5 ms = 200 ms
' P3 Pressed:  3 kHz beep for 1/10 second. 3 kHz = 3000 Hz.
'              1/10 s =  1000 / 10 ms = 100 ms
'{$STAMP BS2}
'{$PBASIC 2.5}
DEBUG "Program Running!"


DO
  IF (IN4 = 1) THEN
    FREQOUT 9, 200, 2000                ' 2000 Hz for 200 ms
  ELSEIF (IN3 = 1) THEN
    FREQOUT 9, 100, 3000                ' 3000 Hz for 100 ms
  ENDIF
LOOP
```

Weiterführende Bücher

"Applied Sensors", Student Guide, Version 2.0, Parallax Inc., 2003

Weitere Soundeffekte und Geräusche mit dem Piezo Schallwandler werden hier vorgestellt. Die pentatonische und wohltemperierte Notenskala bilden die Basis für die Betrachtung mathematischer Brüche. Der Lautsprecher wird als Rückmeldesystem für eine Reihe von Sensormessungen eingesetzt.

"Basic Analog and Digital", Student Guide, Version 2.0, Parallax Inc., 2003

Über einen Lautsprecher wird ein von einem 555-Timerbaustein erzeugter Ton bestimmter Frequenz ausgegeben. Mit der BASIC Stamp wird die Tonfrequenz über den **COUNT** Befehl bestimmt.

***"Understanding Signals"*, Student Guide, Version 1.0, Parallax Inc., 2003**

Mit Hilfe dieses Buches lassen sich die über den `FREQOUT` Befehl erzeugten Töne als digitale Pulsfolge oder als Sinuswelle betrachten.

Kapitel 9: Integrierte Schaltungen

GEHÄUSEFORMEN

Auf dem BASIC Stamp Modul befinden sich verschiedene Integrierte Schaltkreise (IC), die in kleinen schwarzen quaderförmigen Gehäusen (siehe Abbildung 9-1)untergebracht sind. Von der Vielzahl unterschiedlicher Gehäuseformen, die je nach Ausführung mit unterschiedlichen Namen belegt sind, seien hier einige genannt: DIL, SIL, PLCC, SOL usw.. Jedes IC auf dem Modul übernimmt eine spezielle Aufgabe. Oben rechts befindet sich zum Beispiel ein Spannungsregler, der die Batteriespannung auf exakt 5,0 Volt, der Betriebsspannung für alle auf dem Modul befindlichen Bauteile, einregelt. Oben links befindet sich das EEPROM des BASIC Stamp Moduls. PBASIC Programme werden in einem Speicher als sogenannte Token, das sind in Hexadezimalzahlen konvertierte Befehle, verwaltet, die dann im EEPROM des BASIC Stamp Moduls abgelegt werden. Über den BASIC Stamp Editor lassen sich die Token über *Run* und *Memory Map* auf dem Bildschirm anzeigen Der größte Chip ist der sogenannte Interpreter. In ihm werden die aus dem EEPROM abgerufenen Token in PBASIC Handlungsanweisungen übersetzt und anschließend nach dem Prinzip des „Holen und Ausführen“ nacheinander abgearbeitet.

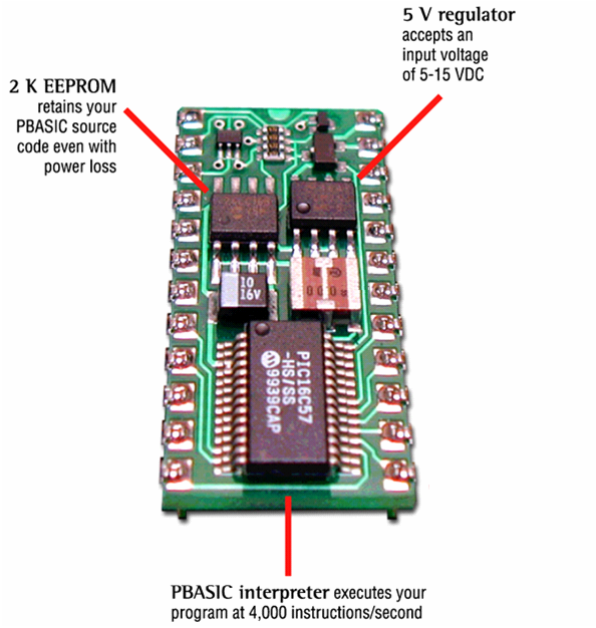


Abbildung 9-1
Integrierte Schaltkreise auf einer BASIC Stamp

Unter einem integrierten Schaltkreis (IC) versteht man im allgemeinen einen kleinen Siliconchip, der in einem schwarzen Plastik- oder Keramikgehäuse untergebracht ist. Je nach Baustein, enthalten integrierte Schaltungen zwischen mehreren hundert bis zu mehreren Millionen Transistoren. Sie sind die Grundbausteine, aus denen integrierte Schaltungen aufgebaut werden. In diesem Kapitel wirst du mit einem Transistor experimentieren. Weitere Komponenten, die in Siliconchips verarbeitet werden, sind Dioden, Widerstände und Kapazitäten.

Lass uns einen Moment innehalten und darüber nachdenken, welche Übungen du bereits ausgeführt hast. Die Liste umfasst: LEDs ein- und ausschalten, Tasterzustände abfragen, Servos steuern, Potentiometereinstellungen auslesen, Lichtmessungen durchführen, steuern von Anzeigen und das Erzeugen von Tönen und Melodien. Obwohl wir erst am Anfang stehen, ist die Aufzählungsliste bereits beeindruckend, besonders dann, wenn du dir klar machst, dass sich alle Übungen irgendwie miteinander kombinieren lassen und damit komplexere Spiele(reien) und Geräte entstehen. Der Kern des Systems, der alle bisher vorgestellten Übungen ermöglichte, setzt sich aus nur drei integrierten Schaltungen und einigen zusätzlichen Teilen zusammen (siehe Abbildung 9-1). Damit

soll dir gezeigt werden wie effizient integrierte Schaltungen sein können, wenn ihre einzelnen Komponenten aufeinander abgestimmt sind und eng zusammenarbeiten.

PROJEKTE MIT INTEGRIERTEN SCHALTKREISEN ERWEITERN

Es gibt tausende integerierter Schaltungen, die sich mit Mikrocontrollern kombinieren lassen. Manchmal gibt es von unterschiedlichen Herstellern integrierte Schaltkreise mit identischen Funktionen, manchmal unterscheiden sie sich geringfügig in ihren Eigenschaften und manchmal unterscheiden sie sich bei völlig identischen Eigenschaften nur im Preis. Jedes dieser ICs lässt sich als Baustein für unterschiedlichste Entwicklungsideen einsetzen. Die Hersteller veröffentlichen in sogenannten Datenblättern, die auch über das World Wide Web herunterzuladen sind, Informationen über die integrierten Schaltkreise ihrer Produktionspalette. In Anwendungsbeispielen, sog. Application notes, stellen sie Schaltungen mit ihren ICs vor, die dem Entwickler bei der Produktentwicklung neue Wege aufzeigen sollen. Dies geschieht in der Hoffnung, dass die Ingenieure den angebotenen Chip auch in ihre neuesten Produktentwicklungen integrieren und somit über den Verkauf des Produktes auch der Verkauf der integrierten Schaltungen angekurbelt wird.

In diesem Kapitel experimentieren wir mit einem Transistor und einem speziellen IC, einem digitalen Potentiometer. Wie bereits weiter oben erwähnt, bilden die Transistoren die Grundbausteine integrierter Schaltungen. Er ist aber auch ein Grundbaustein für eine Vielzahl anderer Schaltkreise. Der Anwendungsbereich eines digitalen Potentiometers ist sehr vielfältig. Erinnern wir uns daran, dass es zu jeder Übung aus den vorangegangenen Kapiteln eine Vielzahl von Einsatzmöglichkeiten dieser integrierten Schaltungen gab.

ÜBUNG 1: STEUERUNG DES STROMFLUSSES MIT EINEM TRANSISTOR

In dieser Übung wird mit Hilfe eines Transistors der Stromfluss durch eine LED gesteuert. Die LED ist als Nachweisgerät gut einsetzbar, da sie bei starkem Stromfluss sehr hell, bei schwachem dagegen nur schwach leuchtet.

Einführung Transistor

Abbildung 9-2 zeigt das Schaltzeichen und die Bauteilzeichnung eines Transistors vom Typ 2N3904. Es gibt eine Vielzahl verschiedener Transistortypen. Dieser sog. NPN Typ erhält seinen Namen vom Aufbau des Transistor und dem verwendeten Material. Seine Arbeitsweise lässt sich mit der eines Strömungsventils vergleichen. Der hier benutzte

Transistor steuert den Strom vom Kollektor C zum Emitter E ausschließlich über einen Steuerstrom in der Basis B. Er ist rund 400 mal geringer als der Strom von C nach E.

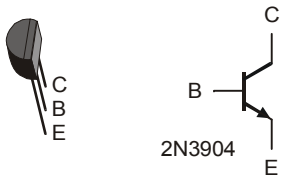


Abbildung 9-2
2N3904 Transistor

Das Datenblatt des Transistors 2N3904: Wie weiter oben bereits erwähnt, veröffentlichen Halbleiterhersteller sogenannte Datenblätter zu ihren Produkten. In ihnen erhält der Entwicklungsingenieur Produktinformationen. Ein Beispiel für einen Auszug aus einem Datenblatt für den 2N3904 findet sich unter: www.fairchildsemi.com. Gib in das Suchfeld ein: 2N3904 und klicke auf *Go*. In einem Suchergebnis sollte sich ein Verweis auf den Produktordner des 2N3904 finden. Folge dem Verweis und lade das Datenblatt über einen Klick auf *Download* herunter. Viele Webbrowser öffnen und zeigen das Datenblatt auch über den Adobe Acrobat Reader.

Bauteile zur Transistorschaltung

1 Transistor – 2N3904
2 Widerstände – 100 kΩ (braun-schwarz-gelb)
1 LED – beliebige Farbe
1 Potentiometer – 10 kΩ
3 Verbindungsdrähte

Aufbau und Test der Transistorschaltung

Abbildung 9-3 zeigt eine Schaltung, mit der manuell der Stromfluss durch eine LED eingestellt wird. Über das Potentiometer wird der Stromfluss zur Basis des Transistors verändert und damit der Stromfluss vom Kollektor zum Emitter gesteuert. Die LED zeigt durch ihr unterschiedlich helles Leuchten die Wirkung der Potentiometereinstellung an.

- √ Baue die Schaltung nach Abbildung 9-3 auf.
- √ Drehe am Potentiometer und überzeuge dich, dass die LED, je nach Potentiometereinstellung ihre Helligkeit verändert.

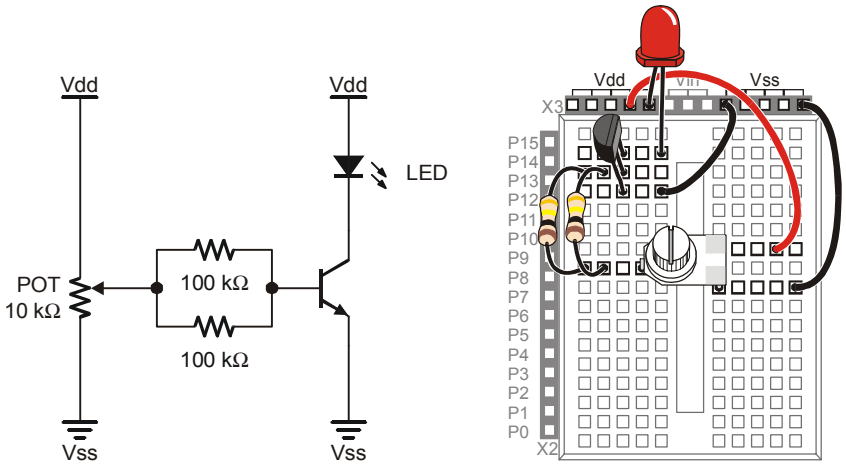


Abbildung 9-3
Manuelle Potentiometer-steuerung einer Transistorschaltung

Du bist dran – Der Transistor als Schalter

Mit der Schaltung aus Abbildung 9-4 wird ein Transistor ein- bzw. ausgeschaltet, er arbeitet als Schalter. Sobald die BASIC Stamp ein HIGH-Signal an die Basis legt, wird der Transistor durchgeschaltet, die LED leuchtet mit maximaler Helligkeit; ein LOW-Signal sperrt den Transistor und die LED erlischt.

Worin besteht der Unterschied zwischen den beiden vorgestellten Schaltungen? Der Stromfluss durch die I/O Pins einer BASIC Stamp ist begrenzt. Auch Transistoren haben Grenzbereiche, die aber deutlich höher liegen. Im *Industrial Control Student Guide*, wird mit Hilfe eines Transistors ein kleiner DC Lüfter angetrieben. Ein Transistor wird dort auch eingesetzt, um einen kleinen Widerstand als Heizelement einzusetzen. Beide Anwendungen benötigen so hohe Stromstärken, dass eine BASIC Stamp sofort zerstört würde, während ein Transistor damit gut fertig wird.

√ Baue die Schaltung nach Abbildung 9-4 auf.

√ Schreibe ein Programm, das zweimal pro Sekunde HIGH und LOW Signale an Pin P8 sendet. Hinweis: **Programmbeispiel:** LedOnOff.bs2 aus Kapitel2 muss nur so verändert werden, dass die HIGH/LOW Signale statt an P3 an P8 ausgegeben werden. Vergiss nicht, das Programm vorher unter einem neuen Namen abzuspeichern, bevor Änderungen vorgenommen werden.

√ Führe das Programm aus und überprüfe, ob die LED ein-/ausgeschaltet werden kann.

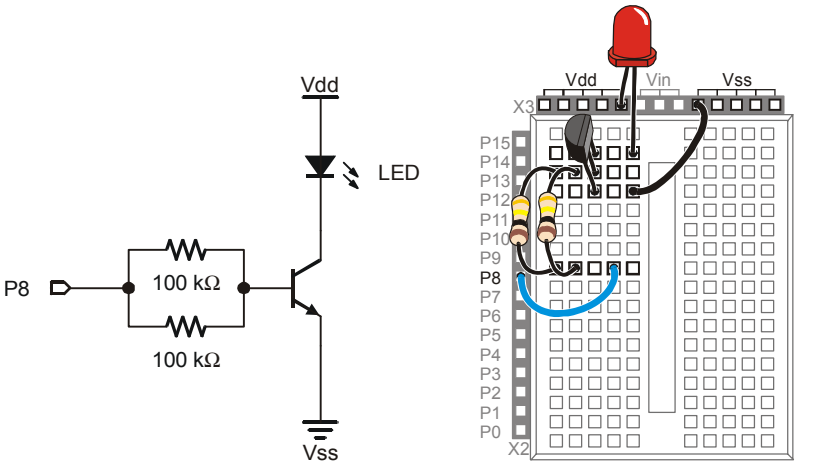


Abbildung 9-4
Schaltung mit BASIC Stamp Steuerung eines Transistorschalters

ÜBUNG 2: SCHALTUNG MIT DIGITALPOTENTIOMETER

In dieser Übung wird das manuell bediente Potentiometer durch ein digital einzustellendes ersetzt. Die BASIC Stamp wird so programmiert, dass sie das digitale Potentiometer und damit auch die Helligkeit der LED, so wie wir es bei dem manuell bedienten Potentiometer in der vorherigen Übung kennengelernt haben, steuert.

Vorstellung eines digitalen Potentiometers

Abbildung 9-5 zeigt die Pinbelegung eines Digitalpotentiometers, so wie es in dieser Übung eingesetzt wird. Es verfügt über 8 Pins, vier auf jeder Seite, mit einem Abstand von je 1/10 inch. Die Hersteller fügen auf dem Plastikgehäuse eine Referenzkerbe ein, so dass die Pinbelegung eindeutig wird. Gezählt wird gegen der Uhrzeigersinn und von der Kerbe an aufwärts.

Ersatzbauteile: Manchmal ist es für Parallax notwendig, auf Ersatzbauteile zurückzugreifen. Diese Bauteile arbeiten genau so exakt, nur die Beschriftung kann unterschiedlich sein. Ist das Gehäuse deines Digitalpotentiometers im Bausatz von *Wie arbeitet ein Mikrocontroller* nicht mit AD5220 beschriftet so besteht kein Grund zur Beunruhigung; es arbeitet auf die gleiche beschriebene Weise und wird in dieser Übung korrekt funktionieren.

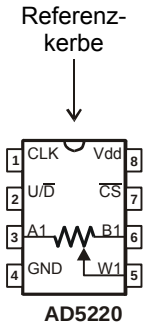


Abbildung 9-5
AD5220 Pinbelegung

Orientiere dich an der Referenzkerbe, um sicherzustellen, dass der AD5220 korrekt in die Schaltung eingebaut wurde.

Hier eine Übersicht der Pins und Funktionen des AD5220:

1. CLK – erhält Clockimpulse (LOW-HIGH-LOW-Signale) zur Bewegung des Schleifers.
2. U/D – Der Pin, der mit einem HIGH Signal den Schleifer (W1) in Richtung A1 bewegt und mit einem LOW Signal in Richtung B1. Dieser Pin bestimmt die Bewegungsrichtung des Schleifers und wird nur aktiv, wenn vorher eine Pulsfolge (LOW-HIGH-LOW) an CLK gesandt wird.
3. A1 – Anschluss A des Potentiometers.
4. GND – Masseverbindung. Der Masseanschluss auf dem Board of Education und BASIC Stamp HomeWork Board ist der Anschluss Vss.
5. W1 – Schleifer des Potentiometers.
6. B1 – Anschluss B des Potentiometers.
7. CS – Chip select Pin. HIGH an diesem Pin ignoriert alle nachfolgenden Steuersignale an CLK und U/D. LOW hebt die Wirkung von HIGH wieder auf.
8. Vdd – Anschluss an +5 V, entspricht Vdd auf dem Board of Education und BASIC Stamp HomeWork Board.

Datenblatt des AD5220: Das Datenblatt des AD5220 kann unter: www.analog.com eingesehen werden. Gib im Suchfeld AD5220 ein und aktiviere *Search*. Klicke auf *Data Sheets link* und "AD5220: Increment/Decrement Digital Potentiometer Datasheet".

Bauteile für ein mit einem Digitalpotentiometer gesteuerten Transistor

1 Transistor – 2N3904
2 Widerstände – 100 kΩ (braun-schwarz-gelb)

1 LED – beliebige Farbe
1 Digitalpotentiometer – AD5220

Aufbau einer Schaltung mit Digitalpotentiometer

Abbildung 9-6 zeigt den Schaltungsaufbau mit einem Digitalpotentiometer statt mit einem manuell betriebenen Potentiometer. Abbildung 9-7 zeigt einen zugehörigen Verdrahtungsplan der Schaltung. Die BASIC Stamp steuert das Digitalpotentiometer über Steuersignale an den Pins P5 und P6

√ Baue die Schaltung nach Abbildung 9-6 und Abbildung 9-7 auf.

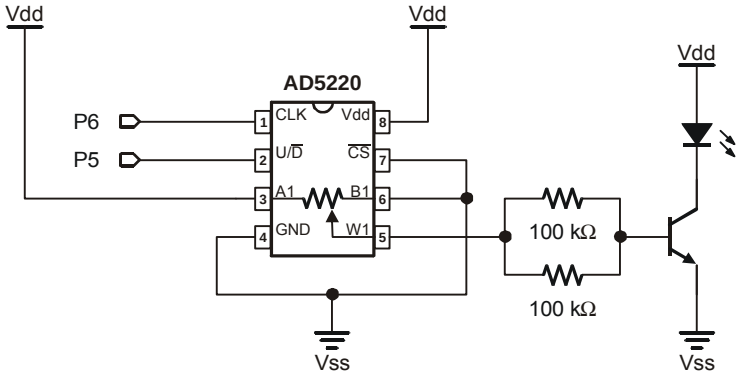


Abbildung 9-6
Schaltungsaufbau
Mit einem
Digitalpotentiometer
gesteuerte Transistor-
schaltung

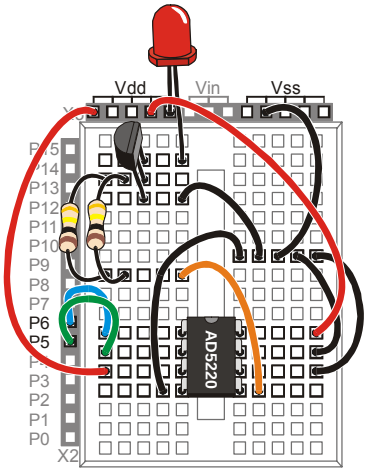


Abbildung 9-7
Verdrahtungsplan
Mit einem
Digitalpotentiometer
gesteuerte
Transistor-
schaltung

Programmierung: Steuerung eines Digitalpotentiometers

Stellen wir uns vor, das der Schleifer des manuell betriebenen Potentiometers aus der letzten Übung über 128 Einstellpositionen verfügt. Stellen wir uns ebenfalls vor, dass der Schleifer in Mittelstellung ist. Das heisst, dass der Schleifer in die eine Richtung 63 Positionen und in die andere 64 Positionen durchlaufen kann.

Nehmen wir nun an, dass der Schleifer um eine Position in Uhrzeigerrichtung gedreht wird. Die LED wird daraufhin ein ganz klein wenig heller leuchten. Dies wäre damit vergleichbar, ein HIGH Signal an den U/D Pin des AD5220 zu legen und eine HIGH-LOW-HIGH Pulsfolge an den CLK Pin zu übertragen.

```
HIGH 5
PULSOUT 6, 1
```

Stellen wir uns jetzt vor, dass dass Potentiometer manuell um 3 Schritte gegen den Uhrzeigersinn gedreht wird. Die LED wird schwächer leuchten. Dies ist wieder vergleichbar mit einem LOW Signal am U/D Pin des AD5220 gefolgt von drei Pulsfolgen an den CLK Pin.

```
LOW 5
FOR counter = 1 TO 3
  PULSOUT 6, 1
  PAUSE 1
NEXT
```


Im nächsten Schritt soll das Potentiometer in Uhrzeigerrichtung ganz nach rechts gedreht werden. Das ist vergleichbar mit einem HIGH Signal am U/D Pin des AD5220 und 65 Pulsfolgen an den CLK Pin. Die LED sollte jetzt hell aufleuchten.

```
HIGH 5
FOR counter = 1 TO 65
  PULSOUT 6, 1
  PAUSE 1
NEXT
```

Zum Abschluss wird das manuelle Potentiometer ganz nach links (gegen den Uhrzeigersinn) gedreht. Die LED geht aus. Dies ist vergleichbar mit einem LOW Signal am U/D Pin und 128 Pulsfolgen an den CLK Pin

```
LOW 5
FOR counter = 0 TO 127
  PULSOUT 6, 1
  PAUSE 1
NEXT
```

Programmbeispiel: DigitalPotUpDown.bs2

In diesem Beispielprogramm durchläuft der Schleifer des Potentiometers alle Positionen vom linken Anschlag bis zum rechten Anschlag; die LED leuchtet hell auf und erlischt.

√ Führe das Programm DigitalPotUpDown.bs2 aus.

```
' What's a Microcontroller - DigitalPotUpDown.bs2
' Sweep digital pot through values.

' {$STAMP BS2}
' {$PBASIC 2.5}

DEBUG "Program Running!"

counter        VAR     Byte

DO

  LOW 5

  FOR counter = 0 TO 127
    PULSOUT 6, 1
    PAUSE 10
  NEXT

  HIGH 5

  FOR counter = 0 TO 127
    PULSOUT 6, 1
    PAUSE 10
  NEXT

LOOP
```

Du bist dran – Die Einstellgeschwindigkeit ändern und den Programmiercode kürzen

Die Geschwindigkeit, mit der sich die Helligkeit einer LED verändern lässt, kann über das ***Duration*** Argument im Befehl **PAUSE** erhöht oder erniedrigt werden.

√ Veränder das Programm durch Einfügen des Befehls **PAUSE 20** und starte es neu. Beobachte die unterschiedliche Geschwindigkeit bei der Helligkeitsänderung.

√ Wiederhole den ersten Schritt mit **PAUSE 5**.

Mit dem Befehl **TOGGLE** lässt sich das Programm übersichtlicher gestalten. **TOGGLE** verändert den Status eines BASIC Stamp I/O Pins. Wurde an den I/O Pin ein HIGH Signal angelegt, macht **TOGGLE** daraus ein LOW Signal und umgekehrt.

- √ Speicher DigitalPotUpDown.bs2 unter DigitalPotUpDownWithToggle.bs2 ab.
- √ Veränder das Programm so, dass es wie das unten dargestellte aussieht.
- √ Führe das Programm aus und überprüfe, ob es genau so arbeitet wie DigitalPotUpDown.bs2.
- √ Vergleiche die Anzahl der Programmzeilen beider Programme.

Speicherüberlauf ist ein Problem, das immer dann auftaucht, wenn ein BASIC Stamp Projekt sehr umfangreich und kompliziert wird. Mit **TOGGLE,** statt mit zwei **FOR…NEXT** Schleifen, haben wir ein Beispiel von vielen kennengelernt, bei dem die gleiche Arbeit mit nur halb so viel Speicherbedarf getan wird.

```
' What's a Microcontroller - DigitalPotUpDownWithToggle.bs2
' Sweep digital pot through values.

' {$STAMP BS2}
' {$PBASIC 2.5}
DEBUG "Program Running!"

counter        VAR     Byte

LOW 5

DO
  FOR counter = 0 TO 127
    PULSOUT 6,5
    PAUSE 10
  NEXT
  TOGGLE 5
LOOP
```


Blick in ein Digitalpotentiometer

Abbildung 9-8 zeigt das Diagramm des Potentiometers in einem AD5220. Dieser Baustein verfügt über 128 Widerstandselemente, von denen jedes einen Wert von 78.125 Ω (Nennwert) hat. Zusammen ergeben die 128 Elemente einen Gesamtwert von 10,000 Ω oder 10 kΩ.

Ein Nennwert ist ein mittlerer Wert. Bauteile wie Widerstände und Kondensatoren werden mit ihrem Nennwert und einer Toleranz angegeben. Jedes Widerstandselement des AD5220 hat einen Nennwert von 78.125 Ω mit einer Toleranz von 30% (23.438 Ω) über oder unter dem Nennwert.

Zwischen jedem der Widerstandselemente befindet sich ein Schalter (Tap). Jeder Schalter ist aus einer Gruppe von Transistoren aufgebaut, die ein- bzw. ausgeschaltet werden und dadurch den Strom leiten oder sperren. Es kann immer nur ein Schalter zur Zeit geschlossen werden. Ist einer der oberen Schalter geschlossen (Pos. 125, 126, oder 127), wirkt das so, als ob ein manuelles Potentiometer ganz nach rechts gedreht wurde. Wird Schalter 0 oder 1 geschlossen, entspricht das einem Dreh in die ganz linke Position.

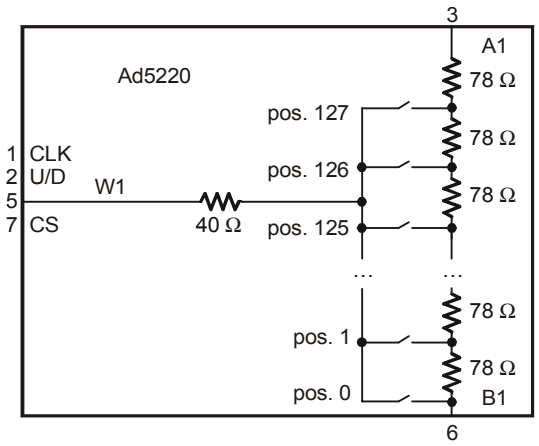


Abbildung 9-8
Innenaufbau des AD5220

Nimm an, das Schalter 126 geschlossen ist. Soll in einem nächsten Schritt Schalter 125 geschlossen werden, (öffne Pos. 126 und schließe Pos. 125), setze U/D auf LOW und schicke einen Puls an CLK. Soll Tap 127 geschlossen werden, setze U/D auf HIGH und schicke zwei Pulse an CLK. Tap 1 wird geschlossen, wenn U/D auf LOW geht und 126 Pulse an CLK folgen.

Im folgenden Beispielprogramm wird über das Debug Terminal abgefragt, welches Tap gesetzt werden soll. Dann entscheidet das Programm, ob der U/D Pin auf HIGH oder LOW gesetzt wird und schickt anschließend die entsprechende Anzahl Pulse an CLK, um das Tap von der alten Stellung in die neue zu überführen.

Mit Ausnahme der EEPROM Daten, zeigt das nächste Beispielprogramm alle Abschnitte, die auch in einem Anwendungsprogramm normalerweise zu finden sind:

- Title – Kommentar, in dem der Dateiname des Programms, eine kurze Beschreibung und die Stamp und PBASIC Direktiven erscheinen.
- EEPROM Data –**DATA** Deklarationen, die vom Programm genutzt werden
- I/O Definitions – Konstantendeklarationen der I/O Pins
- Constants – Konstantendeklarationen
- Variables – Variablendeklarationen
- Initialization – Initialisierungsroutine, in der Startbedingungen festgelegt werden. Im folgenden Programm wird der Potentiometerschleifer ganz nach links gedreht.
- Main – Hauptprogramm, von dem aus alles gesteuert wird.
- Subroutines – Unterprogramme, in denen Teilaufgaben abgearbeitet werden.

Programmbeispiel: TerminalControlledDigtialPot.bs2

Benutze dieses Beispielprogramm und das Debug Terminal um das Tap des Digitalpotentiometers zu setzen. Wird die Tap-Einstellung verändert, ändert sich auch die Leuchtstärke der LED. Abbildung 9-9 zeigt, dass während des Programmlaufs der Wert 120 in das Fenster Senden eingegeben wurde. Da die Voreinstellung des Taps 65 war, leuchtet die LED jetzt ungefähr zweimal so hell.

Darstellungsfenster

Senden →

Empfangen →

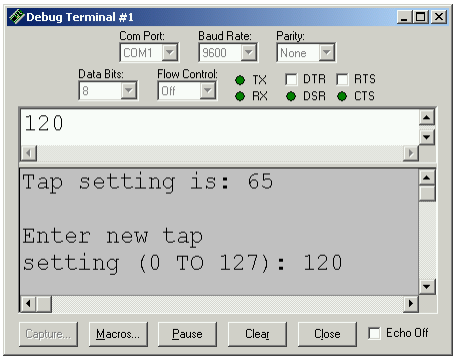


Abbildung 9-9
Mitteilungen an die BASIC Stamp übertragen

Klicke ins Darstellungsfenster Senden(oben) und gib die Zahl für das neue Tap ein.

√ Führe das Programm TerminalControlledDigtialPot.bs2 aus.
√ Gib Werte zwischen 0 und 127 über das Debug Terminal ein. Schließe deine Eingabe mit der Eingabetaste ab.

```
' -----[ Title ]-----------------------------------------------------------
' What's a Microcontroller - TerminalControlledDigitalPot.bs2
' Update digital pot's tap based on Debug Terminal user input.

' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ EEPROM Data ]-------------------------------------------------


' -----[ I/O Definitions ]---------------------------------------------

UdPin          CON     5                    ' U/D an Pin P5
ClkPin         CON     6                    ' CLK an Pin P6

' -----[ Constants ]--------------------------------------------------

DelayPulses    CON     10                   ' Verzögerung für LED Beob.
DelayReader    CON     2000

' -----[ Variables ]--------------------------------------------------

counter        VAR     Byte                 ' Zähler in FOR...NEXT Schleife
oldTapSetting  VAR     Byte                 ' alte Tap Einstellung
newTapSetting  VAR     Byte                 ' neue Tap Einstellung

' -----[ Initialization ]---------------------------------------------

oldTapSetting = 0                           ' Taps alt und neu bei Start
newTapSetting = 0                           ' auf Null setzen

LOW UdPin                                   ' Setze U/D Pin auf LOW.
FOR counter = 0 TO 127                      ' Setze Tap auf kleinste Pos.
  PULSOUT 6,5
  PAUSE 1
NEXT

' -----[ Main Routine ]-----------------------------------------------

DO

  GOSUB Get_New_Tap_Setting                 ' Subroutine mit Abfrage
  GOSUB Set_Ud_Pin                          ' Setze U/D Pin auf LOW/HIGH
  GOSUB Pulse_Clk_pin                       ' Pulsfolge liefern.

LOOP

' -----[ Subroutines ]------------------------------------------------

Get_New_Tap_Setting:                        ' Displayanzeige mit Eingabe-
                                            ' aufforderung für neue
```

```
  DEBUG CLS, "Tap setting is: ",              ' Tap Einstellungen.
        DEC newTapSetting, CR, CR
  DEBUG "Enter new tap", CR, "setting (0 TO 127): "
  DEBUGIN DEC newTapSetting

  RETURN

Set_Ud_Pin:                                   ' Berechne neue + alte Tapwerte
                                              ' und lege U/D Pinwert fest.
  IF newTapSetting > oldTapSetting THEN       ' Informiere Nutzer, wenn Werte
    HIGH UdPin                                ' identisch.
    oldTapSetting = oldTapSetting + 1         ' Erhöhung an Pulse_Clk_pin.
  ELSEIF newTapSetting < oldTapSetting THEN
    LOW UdPin
    oldTapSetting = oldTapSetting - 1         ' Erniedrigung an Pulse_Clk_pin.

  ELSE
    DEBUG CR, "New and old settings", CR,
              "are the same, try ", CR,
              "again...", CR
    PAUSE DelayReader                         ' Zeitschleife für Leser-
  ENDIF                                       ' information.

  RETURN

Pulse_Clk_pin:

  ' Deliver pulses from old to new values.  Keep in mind that Set_Ud_Pin
  ' adjusted the value of oldTapSetting toward newTapSetting by one.
  ' This keeps the FOR...NEXT loop from executing one too many times.

  FOR counter = oldTapSetting TO newTapSetting
    PULSOUT ClkPin, 1
    PAUSE DelayPulses
  NEXT

  oldTapSetting = newTapSetting               ' oldTapSetting ständig
                                              ' aktualiseren
  RETURN
```

ZUSAMMENFASSUNG

In diesem Kapitel haben wir integrierte Schaltkreise kennengelernt und gezeigt, wie man sie zusammen mit der BASIC Stamp nutzt. Es wurden die Referenzkerbe und die Pinbelegung als wichtige Elemente integrierter Schaltungen angesprochen. Ein Transistor wurde als Strömungsventil eingesetzt und über ein Digitalpotentiometer der Stromfluss durch einen Transistor gesteuert. Die Funktion aller Pins des Digitalpotentiometers wurden angesprochen und die interne Struktur erklärt. **`TOGGLE`** haben wir neu und als speichersparenden PBASIC Befehl kennengelernt

Verständnisfragen

1. Wie heißen die Anschlüsse des Transistors, den wir in diesem Kapitel benutzt haben?
2. Welcher Anschluss steuert den Strom durch den Transistor?
3. Wodurch lässt sich der Strom durch einen Transistor erhöhen oder erniedrigen?

Übung

1. Schreibe ein Programm, das, ausgehend von einer beliebigen aktuellen Einstellung, den Tap eines Digitalpotentiometers in Position 0 schließt.

Projekt

1. Füge dem Projekt einen Fotowiderstand hinzu und passe die Helligkeit der LED an die Helligkeit an, die der Fotowiderstand erkennt.

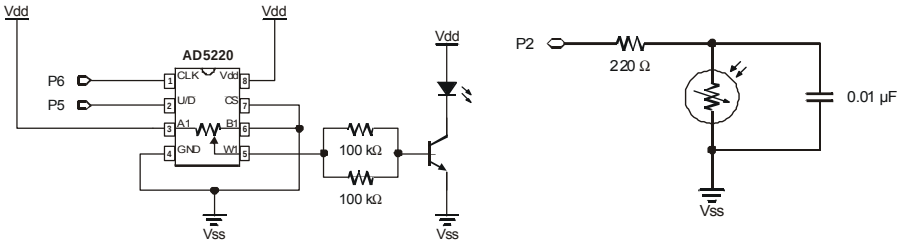
Lösungen

Q1. Emitter, Basis, Kollektor.
Q2. Die Basis steuert den Strom durch den Transistor.
Q3. Über die Basis des Transistors.
E1. Schaue in das Programm TerminalControlledDigitalPot.bs2. Zuallererst muss im Initialisierungsabschnitt das Tap auf die niedrigste Position gesetzt werden. Der Programmcode wird unten angegeben.

```
' What's a Microcontroller - Ch9Ex01_SetTapToZero.bs2
' Turn tap on digital pot all the way down to zero
' {$STAMP BS2}
' {$PBASIC 2.5}
DEBUG "Program Running!"
```

```
UdPin           CON     5               ' Set values of I/O pins
ClkPin          CON     6               ' connected to CLK and U/D.
counter         VAR     Byte                ' Counter for FOR...NEXT.

LOW UdPin                               ' Set U/D pin for Down.
FOR counter = 0 TO 128                  ' Set tap to lowest position.
  PULSOUT ClkPin,5
  PAUSE 1
NEXT
```

P1. Nimm die Schaltung Digitalpotentiometer aus Abbildung 9-6, S. 288 und die Schaltung Fotowiderstand aus Abbildung 7-2, S. 203.



Die Lösung basiert auf dem Programm TerminalControlledDigitalPot.bs2, mit der Änderung, dass die Eingabe nicht über das Debug Terminal erfolgt, sondern über das Auslesen des Fotowiderstandes. Das Unterprogramm **`Read_Photoresistor`**, liest den Fotowiderstand über **`RCTIME`** aus, wie es in Kapitel 5 gezeigt wurde. Dieser Wert wird die neue Tapeinstellung und das Originalprogramm setzt den Tap. Die Anteile des Originalprogramms, in denen die Eingaben abgefragt werden, wurden gelöscht, inklusive des Unterprogramms **`Get_New_Tap_Setting`**.

```
' What's a Microcontroller - Ch9Prj01_PhotoControlledDigitalPot.bs2
' Corrected version of program on page 277 of v2.2 text.
' Update digital pot's tap based on photoresistor reading
' {$STAMP BS2}
' {$PBASIC 2.5}
DEBUG "Program Running!"

' -----[ Declarations and Initialization]------------------------------
UdPin           CON     5               ' Set values of I/O pins
ClkPin          CON     6               ' connected to CLK and U/D.
PhotoPin        CON     2               ' Photoresistor on pin P2
```

```
DelayPulses    CON     10              ' Delay to observe LED fade.
DelayReader    CON     2000

counter        VAR     Byte            ' Counter for FOR...NEXT.
oldTapSetting  VAR     Byte            ' Previous tap setting.
newTapSetting  VAR     Byte            ' New tap setting.
lightReading   VAR     Word            ' reading from photoresistor

oldTapSetting = 0                      ' Initialize new and old
newTapSetting = 0                      ' tap settings to zero.

LOW UdPin                              ' Set U/D pin for Down.
FOR counter = 0 TO 128                 ' Set tap to lowest position.
  PULSOUT 6,5
  PAUSE 1
NEXT

' -----[ Main Routine ]------------------------------------------------
DO
  GOSUB Read_Photoresistor
  LightReading = lightReading MAX 127   ' Constrain to max of 127
  newTapSetting = lightReading
  GOSUB Set_Ud_Pin                      ' Set U/D pin for up/down.
  IF newTapSetting <> oldTapSetting THEN' No change if light level
steady
    GOSUB Pulse_Clk_pin                 ' Deliver pulses.
  ENDIF
LOOP

' -----[ Subroutines ]-------------------------------------------------
Set_Ud_Pin:                                    ' Examine old and new
  IF newTapSetting > oldTapSetting THEN        ' tap values to decide
    HIGH UdPin
    OldTapSetting = oldTapSetting + 1
  ELSEIF newTapSetting < oldTapSetting THEN    ' value of UdPin. Notify
    LOW UdPin                                  ' user if values are
    OldTapSetting = oldTapSetting - 1
  ENDIF                                        ' equal.
  RETURN

Pulse_Clk_pin:                                  ' Deliver pulses
  FOR counter = oldTapSetting TO newTapSetting ' from old to new
    PULSOUT ClkPin, 1                           ' values.
    PAUSE DelayPulses
  NEXT
  oldTapSetting = newTapSetting         ' Keep track of new and old
  RETURN                                ' tapSetting values.

Read_Photoresistor:
  HIGH PhotoPin
  PAUSE 100
```

```
RCTIME PhotoPin, 1, lightReading
RETURN
```

Weiterführende Bücher

"Industrial Control", Student Guide, Version 2.0, Parallax Inc., 2002

Industrial Control nutzt den Transistor als Ein-/Ausschalter für ein Widerstands-Heizelement. Mit einem Fototransistor wird auch die Umdrehungszahl eines Spinnrades bestimmt.

Kapitel 10: Komplexe Systeme

SUBSYSTEME INTEGRIEREN

Die meisten Übungen in diesem Buch zeigen, wie die BASIC Stamp programmiert werden muss, damit sie mit einem oder zwei Schaltkreisen gleichzeitig arbeiten kann. Viele Mikrocontroller müssen hunderte von Schaltkreisen überwachen und steuern. In diesem Kapitel werden einige Techniken gezeigt, mit denen mehrere Schaltungen über einen Mikrocontroller verwaltet werden können. Einen Mikrocontroller so zu programmieren, dass er die Aktionen verschiedener Schaltungen, von denen jede eine besondere Funktion ausübt, überwacht, nennt man Integration von Subsystemen.

In diesem Kapitel betrachten wir ein Sensorfeld aus zwei Taster-Schaltungen sowie einer Potentiometer- und einer Fotowiderstandsschaltung. In den Übungen werden wir Aufbau und Test jedes Subsystems einzeln durchführen. Ist jedes Subsystem aufgebaut und getestet, wird ein Programm entwickelt, das alle Systeme in einer Steuerungseinheit zusammenfasst. Abbildung 10-1 zeigt den kompletten Schaltungsaufbau, die zugehörige Bauteilliste befindet sich weiter unten.

Wenn möglich, sollte jedes Subsystem einzeln getestet werden, bevor es in eine größere Einheit eingebaut wird. Wird diese Regel beherzigt, lassen sich Projekte reibungsloser umsetzen und die Erfolgschancen für ein lauffähiges System erhöhen sich deutlich. In den Übungen dieses Kapitels wird nach diesem Muster verfahren.

Bauteilliste: Sensorfelder

4 Widerstände – 220 Ω (rot-rot-braun)
2 Widerstände – 10 kΩ (braun-schwarz-orange)
2 Taster als Schließer
2 Kondensatoren – 0.1 µF
1 Potentiometer – 10 kΩ
1 Fotowiderstand
5 Verbindungsdrähte

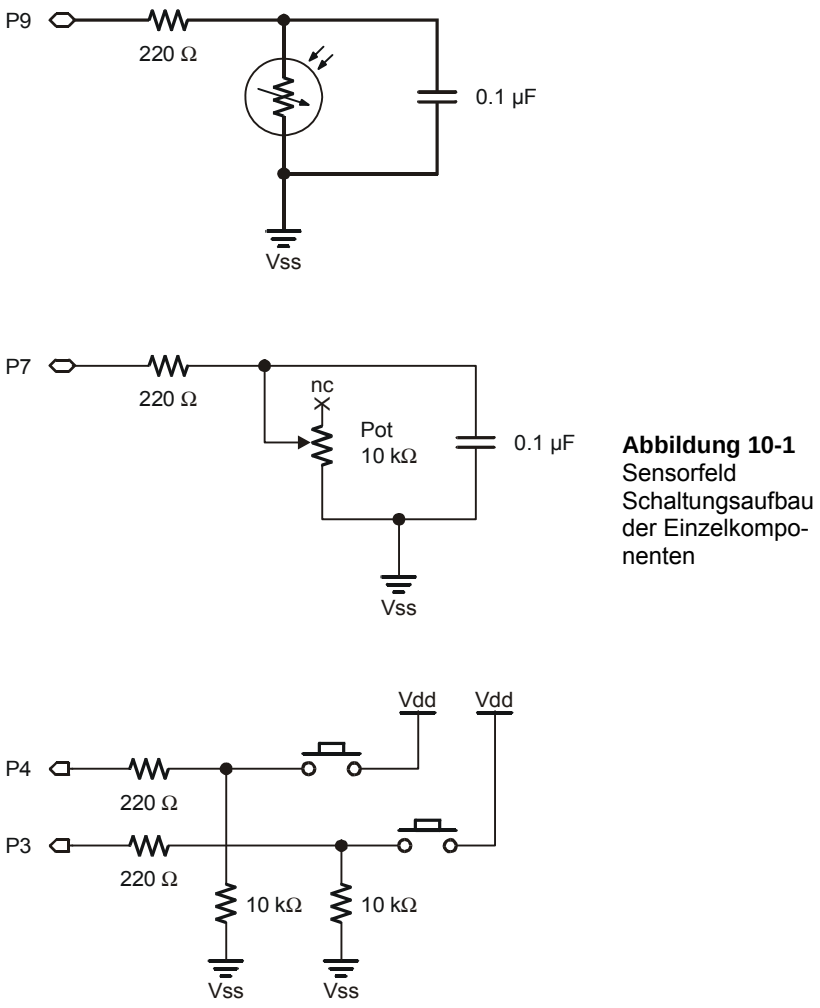


Abbildung 10-1
Sensorfeld Schaltungsaufbau der Einzelkomponenten

ÜBUNG 1: AUFBAU UND TEST VON ZWEI TASTERSCHALTUNGEN

Diese Übung beginnt mit Aufbau und Test einer einzelnen Tasterschaltung. Wenn wir uns überzeugt haben, dass die erste Schaltung fehlerfrei läuft, kann die zweite Tasterschaltung aufgebaut und getestet werden.

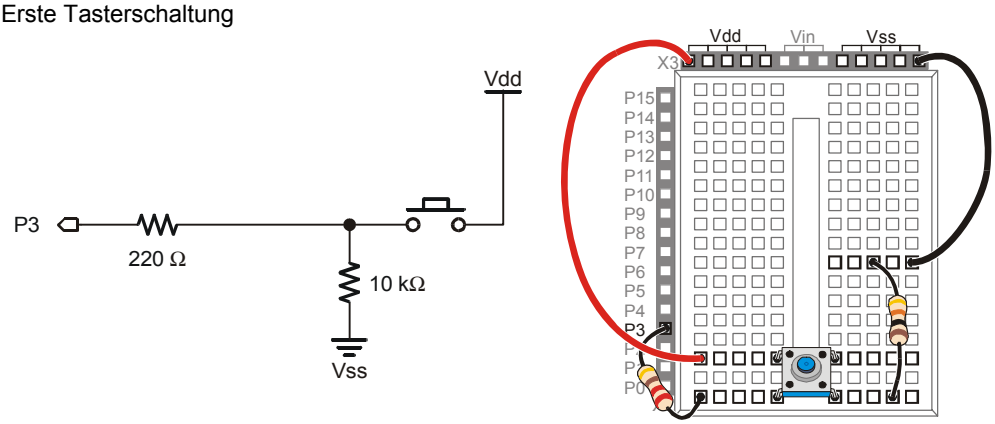
Bauteile Tasterschaltungen

2 Taster als Schließer
2 Widerstände – 10 kΩ (braun-schwarz-orange)
2 Widerstände – 220 Ω (rot-rot-braun)
2 Verbindungsdrähte

Aufbau der ersten Tasterschaltung

Baue die Tasterschaltung nach Abbildung 10-2 auf.

Abbildung 10-2
Erste Tasterschaltung



Programm zur Tasterschaltung

Es wird dir sicher leicht fallen (siehe Kapitel 3) ein Programm zu schreiben, das die Schaltung austestet.

Programmbeispiel: ReadPushbuttonState.bs2

Dies ist ein bereits bekanntes Programm zum Testen von Tastern aus Kapitel 3:

- √ Führe das Programm ReadPushbuttonState.bs2 aus.
- √ Überprüfe, ob die BASIC Stamp den Tasterzustand auslesen kann.
- √ Korrigiere alle auftretenden Fehler, bevor die nächste Schaltung aufgebaut wird.

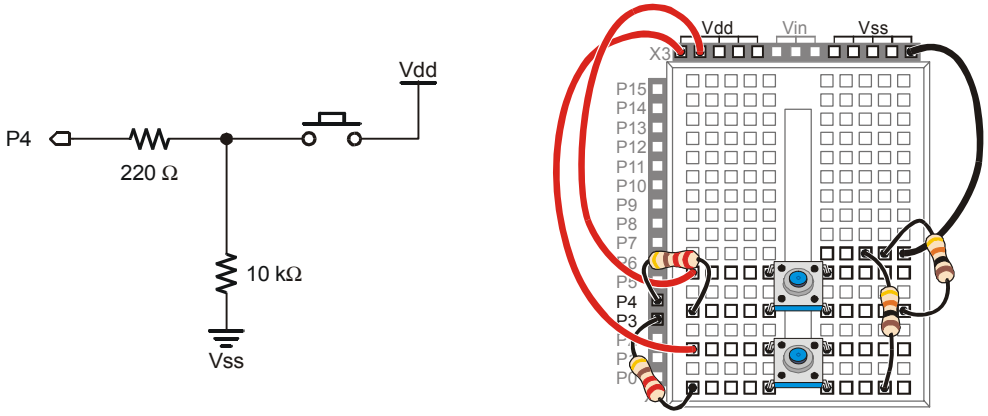
```
' What's a Microcontroller - ReadPushbuttonState.bs2
' Check and send pushbutton state to Debug Terminal every 1/4 second.

' {$STAMP BS2}
' {$PBASIC 2.5}

DO

  DEBUG ? IN3
  PAUSE 250

LOOP
```

– Aufbau und Test der zweiten Tasterschaltung

Nachdem die erste Tasterschaltung aufgebaut und getestet wurde, kann mit der zweiten begonnen werden.

- √ Füge die Tasterschaltung, wie in Abbildung 10-3 dargestellt, hinzu.
- √ Teste die Schaltung, indem das Programm ReadPushbuttonState.bs2 so abgeändert wird, dass es die Teilschaltung, die mit Pin P4 verbunden ist, ausliest. Führe das Programm aus und überprüfe, ob es fehlerfrei läuft.
- √ Korrigiere alle auftretenden Fehler, bevor die nächste Übung begonnen wird.

Abbildung 10-3
Eine zweite Tasterschaltung dem Projekt hinzufügen



ÜBUNG 2: AUFBAU UND TEST VON ZWEI RC-SCHALTUNGEN

Nach den beiden Tasterschaltungen werden wir uns jetzt die RC-Schaltungen vornehmen.

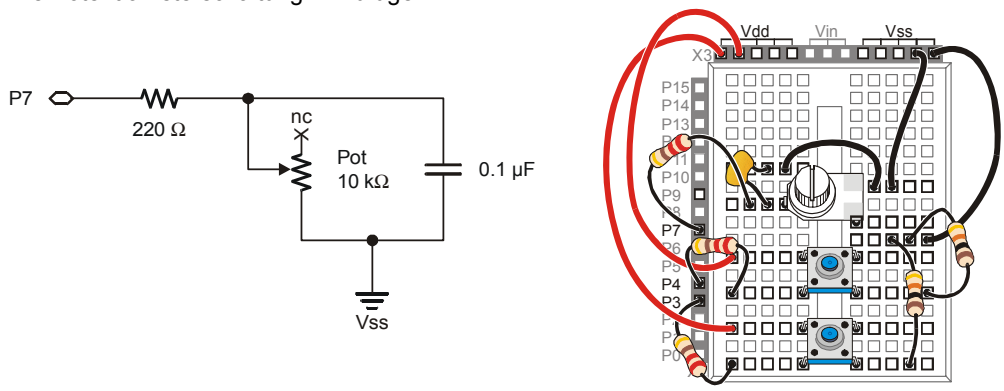
Ergänzungsbauteile für die Potentiometer- und Fotowiderstandsschaltungen

2 Widerstände – 220 Ω (rot-rot-braun)
2 Kondensatoren – 0.1 µF
4 Verbindungsdrähte
1 Fotowiderstand
1 Potentiometer – 10 kΩ

Aufbau der Potentiometerschaltung

√ Füge die Potentiometerschaltung, wie in Abbildung 10-4 gezeigt, in das Projekt ein.

Abbildung 10-4
Eine Potentiometerschaltung hinzufügen



Test der Potentiometerschaltung

Teste diese Schaltung mit dem Programm ReadPotWithRcTime.bs2. Es wurde bereits in Kapitel 5 benutzt, um die Potentiometerschaltung auszulesen.

Programmbeispiel: ReadPotWithRcTime.bs2

√ Führe das Programm ReadPotWithRcTime.bs2 aus.

- √ Überprüfe, ob die BASIC Stamp realistische und verwertbare Messwerte vom Potentiometer empfängt.
- √ Korrigiere mögliche Fehler, bevor die nächste Schaltung aufgebaut wird.

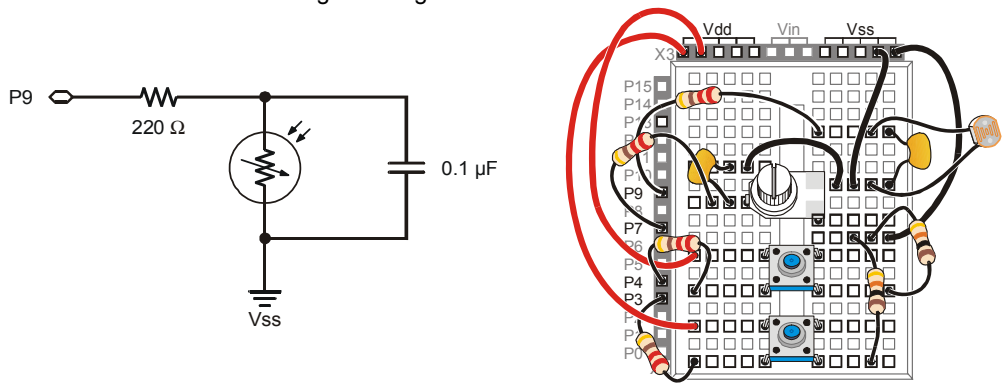
```
' What's a Microcontroller - ReadPotWithRcTime.bs2
' Read potentiometer in RC-time circuit using RCTIME command.

' {$STAMP BS2}
' {$PBASIC 2.5}

time           VAR     Word

DO

  HIGH 7
  PAUSE 100
  RCTIME 7, 1, time
  DEBUG HOME, " time =  ", DEC5 time

LOOP
```

Du bist dran – Aufbau und Test der Fotowiderstandsschaltung

Nachdem die RC-Schaltung aufgebaut und getestet wurde, kann mit der zweiten RC-Schaltung begonnen werden.

- √ Füge die Schaltung mit dem Fotowiderstand zu den bereits bestehenden Schaltungen auf dem Board hinzu, so wie in Abbildung 10-5 gezeigt.
- √ Verändere ReadPotWithRcTime.bs2 so, dass berücksichtigt ist, dass der Fotowiderstand mit P9 verbunden ist.
- √ Korrigiere alle auftretenden Fehler, bevor es zur nächsten Übung geht.

Abbildung 10-5
Eine Fotowiderstandsschaltung hinzufügen



ÜBUNG 3: EIN BEISPIEL FÜR DIE INTEGRATION VON SUBSYSTEMEN

Nachdem nun alle Sensorelemente einzeln aufgebaut und getestet worden sind, schreiben wir ein Programm, das die vier Schaltungen verwaltet. In diesem Beispiel wird gezeigt, wie die BASIC Stamp zu programmieren ist, damit ein Anwender die Sensoren einzeln über eine Menü aufrufen kann.

10

Programmierung eines Menüsystems mit der PIN Direktive

Der Befehl **ON…GOSUB** kann für die Programmierung von Menüs sehr hilfreich sein.

ON *offset* , GOSUB *Target1, {Target2, Target3,…}*

Im folgenden Programmbeispiel nutzt der **ON...GOSUB** Befehl den Wert der Variablen **request,** um damit auf eines von insgesamt vier Unterprogrammen zu verweisen. Ist **request** identisch 0 springt das Programm mit **GOSUB Read_Pushbutton_1** in das Unterprogramm **Read_Pushbutton_1**. Mit **request** identisch 1 wird die Zeile **GOSUB Read_Pushbutton_2** ausgeführt und so weiter.

```
ON request GOSUB Read_Pushbutton_1, Read_Pushbutton_2,
                 Read_Pot, Read_Photoresistor
```

Mit der **PIN** Direktive kann in einem PBASIC Programm jedem I/O Pin ein Name zugewiesen werden. Der BASIC Stamp Editor überprüft, ob der I/O Pin als Eingabe- oder als Ausgabepin (oder beides) genutzt wird. Die Syntax für die **PIN** Direktive lautet:

PinName* PIN *PinNumber

Im folgenden Beispielprogramm wird gezeigt, wie man einen ***PinName*** deklariert und im Programm benutzt. Wenn als ***PinName*** **Pb1Pin** in einem **DEBUG** Befehl benutzt wird, geht der BASIC Stamp Editor davon aus, dass der Eingabewert eines I/O Pins (**IN3**) angezeigt werden soll. Ist der Taster an P3 gedrückt, zeigt der Befehl **DEBUG** eine 1, im nicht gedrückten Zustand eine 0 an. Die Erklärung: Der BASIC Stamp Editor weiß, dass er in **Pb1Pin** den Wert von **IN3** einzusetzen hat.

```
Pb1Pin          PIN     3
DEBUG CLS, ? Pb1Pin
```

Es folgt ein weiteres Beispiel, in dem der BASIC Stamp Editor weiß, dass er den Wert **9** einzusetzen hat, weil der ***PinName*** **PhotoPin** in den Befehlen **HIGH** und **RCTIME** benutzt wird.

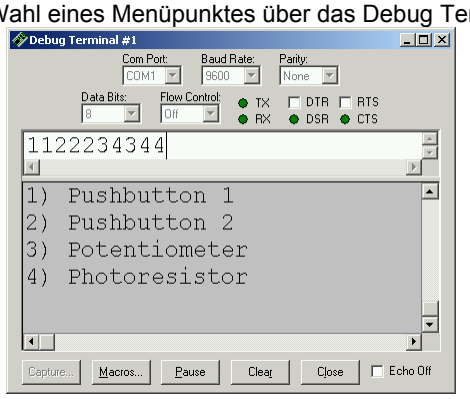
```
PhotoPin        PIN     9
HIGH PhotoPin
RCTIME PhotoPin, 1, time
```

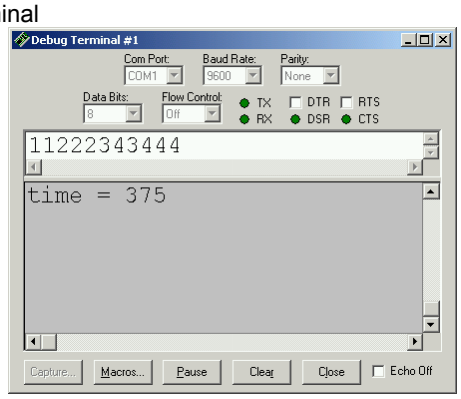
Programmbeispiel: TerminalOperatedSensorArray.bs2

Abbildung 10-6 zeigt das Debug Terminal, nach Aufruf des Programms TerminalOperatedSensorArray.bs2. Im linken Fenster wird das Hauptmenü dargestellt, im rechten wird für das Beispiel, dass der Menüpunkt 4 gewählt wurde, eine mögliche Ausgabe gezeigt. Vergiss bitte nicht, im oberen Sichtfenster durch Eingabe einer Zahl zwischen 1 und 4 eine Auswahl zu treffen und der BASIC Stamp zu übertragen.

- √ Führe das Programm TerminalOperatedSensorArray.bs2 aus.
- √ Klicke in das obere Sichtfenster des Debug Terminals und bestimme durch Eingabe einer ganzen Zahl einen Menüpunkt.
- √ Der angezeigte Messwert wird für 1,5 Sekunden im unteren Fenster angezeigt und wurde zum Zeitpunkt der Menüwahl ermittelt. Dies muss uns bewusst sein, wenn wir eine Auswahl treffen und gleichzeitig vielleicht noch einen Taster gedrückt halten, am Potentiometer schrauben oder unachtsamer Weise den Fotowiderstand beschatten.

Abbildung 10-6
Wahl eines Menüpunktes über das Debug Terminal






```
' -----[ Title ]-----------------------------------------------------------
' What's a Microcontroller - TerminalOperatedSensorArray.bs2
' Use Debug Terminal to choose to read one of four sensors.

' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ I/O Definitions ]--------------------------------------------------

Pb1Pin          PIN     3
Pb2Pin          PIN     4
PotPin          PIN     7
PhotoPin        PIN     9

' -----[ Constants ]-------------------------------------------------------

DelayRc         CON     100
DelayReader     CON     1500

' -----[ Variables ]-------------------------------------------------------

request         VAR     Nib
time            VAR     Word

' -----[ Main Routine ]----------------------------------------------------

DO

  GOSUB Display_Menu
  GOSUB Get_Request
  ON request GOSUB Read_Pushbutton_1, Read_Pushbutton_2,
                   Read_Pot, Read_Photoresistor
```

```
LOOP

' -----[ Subroutine - Display_Menu ]-------------------------------------

Display_Menu:

  DEBUG CLS, "MENU: ", CR, CR,
             "1) Pushbutton 1", CR,
             "2) Pushbutton 2", CR,
             "3) Potentiometer", CR,
             "4) Photoresistor", CR, CR
  RETURN

' -----[ Subroutine - Get_Request ]---------------------------------------

Get_Request:

  DEBUGIN DEC1 request
  request = request - 1

  RETURN

' -----[ Subroutine - Read_Pushbutton_1 ]---------------------------------

Read_Pushbutton_1:

  DEBUG CLS, ? Pb1Pin
  PAUSE DelayReader

  RETURN


' -----[ Subroutine - Read_Pushbutton_2 ]---------------------------------

Read_Pushbutton_2:

  DEBUG CLS, ? Pb2Pin
  PAUSE DelayReader

  RETURN

' -----[ Subroutine - Read_Pot ]------------------------------------------

Read_Pot:

  HIGH PotPin
  PAUSE DelayRc
  RCTIME PotPin, 1, time
  DEBUG CLS, ? time, "     "
  PAUSE DelayReader
```

```
  RETURN

' -----[ Subroutine - Read_Photoresistor ]-------------------------------

Read_Photoresistor:

  HIGH PhotoPin
  PAUSE DelayRc
  RCTIME PhotoPin, 1, time
  DEBUG CLS, ? time, "     "
  PAUSE DelayReader

  RETURN
```

Du bist dran – Unterprogramme anpassen

Der Anwendungsbereich des **ON…GOSUB** Befehl beschränkt sich nicht nur Terminaldarstellungen. Sollen die vier unterschiedlichen Unterprogramme nacheinander aufgerufen werden, dann lässt sich das programmiertechnisch wie folgt umsetzen:

```
DO

  ' GOSUB Display_Menu
  ' GOSUB Get_Request
  FOR request = 0 TO 3
    ON request GOSUB Read_Pushbutton_1, Read_Pushbutton_2,
                     Read_Pot, Read_Photoresistor
  NEXT

LOOP
```

Beachte, dass die beiden Routinen, die sich mit der Terminalausgabe und Darstellung beschäftigen, auskommentiert wurden und das der Befehl **ON…GOSUB** in eine **FOR…NEXT** Schleife eingebettet ist.

- √ Speicher das Beispielprogramm unter dem neuen Namen: TerminalOperatedSensorArrayYourTurn.bs2.
- √ Änder die Hauptroutine so ab, wie oben besprochen.
- √ Führe das Programm aus und überzeuge dich davon, dass es nacheinandern alle Messprogramme aufruft.

ÜBUNG 4: ENTWICKLUNG EINES ZUSÄTZLICHEN SOFTWAREPAKETES

Nehmen wir einmal an, dass das Sensorfeld in einem Gerät untergebracht werden soll, zu dem nur eine ganz bestimmte Gruppe von Menschen einen Passwort geschützten Zugang haben darf. Das heißt, das bestehende Programm muss so erweitert werden, das es Passwörter abspeichert und überprüft. Dazu ist es besser, erst einmal ein eigenständiges Passwortprogramm zu schreiben und es später, wenn alles fehlerfrei arbeitet, in das Hauptprogramm einzubauen.

Programmierung einer Passwortabfrage

Ein Passwort lässt sich in einem PBASIC Programm über die Direktive **DATA** abspeichern. Beispiel:

```
Password        DATA    "pass!"
```

Identitäten: Diese **DATA** Direktive ist identisch mit **Password DATA "p", "a", "s", "s", "!"**.

Es werden zum Abspeichern weiterer Werte einige Variablen im Programm benötigt:

```
index           VAR     Nib
temp            VAR     Byte
```

Zur Einführung eines fünf Zeichen umfassenden Passwortes gibt es eine spezielle Variablendeklaration (sog. Felddeklaration), mit der fünf Variable des gleichen Namens deklariert werden.

```
userEntry       VAR     Byte(5)
```

Damit wurden fünf Variable vom Typ Byte definiert, die alle den Namen userEntry haben: **userEntry(0)**, **userEntry(1)**, **userEntry(2)**, **userEntry(3)** und **userEntry(4)**.

Der Befehl **DEBUGIN** enthält eine Formatanweisung **STR** die automatisch Zeichen in ein Array lädt. Der Aufruf kann zum Beispiel so aussehen:

```
DEBUGIN STR userEntry \5
```

Nach Eingabe von fünf Zeichen in das Fenster *Senden* des Debug Terminals, werden die Zeichen der Reihe nach in den Speichern **userEntry(0)**, **userEntry(1)** usw. abgelegt.

Mit dem PBASIC Schlüsselwort **EXIT** kann eine Schleife sofort verlassen werden. Bei der Passworteingabe kann mit einer Entscheidungsanweisung **IF…THEN** und einem **EXIT** Befehl die Eingabeschleife vorzeitig verlassen werden, wenn nicht alle Eingabezeichen mit dem Passwort identisch sind. Vorzeitig bedeutet hier, dass die Variable **index** nicht bis fünf hochgezählt wird, da das eingegebene Passwort bereits vorher fehlerhaft war:

```
FOR index = 0 TO 4
  READ Password + index, temp
  IF temp <> userEntry(index) THEN EXIT
NEXT

IF index <> 5 THEN
  DEBUG CR,"Password not correct.", CR
ENDIF
```

Im folgenden Beispielprogramm werden der **DEBUGIN** Befehl, die **FOR…NEXT** Schleife und die **IF…THEN** Anweisung in eine **DO…LOOP UNTIL** Schleife eingebettet, die solange durchlaufen wird, bis Index den Wert 5 annimmt, was gleichbedeutend damit ist, dass das eingegebene Passwort korrekt ist.

Programmbeispiel: PasswordChecker.bs2

Abbildung 10-7 zeigt das Debug Terminal bei Programmausführung von PasswordChecker.bs2. Das Programm wartet darauf, dass die Buchstabenfolge „pass!“ eingegeben wird.

- √ Führe das Programm PasswordChecker.bs2 aus.
- √ Gib versuchsweise Buchstabenkombinationen ein, die nicht dem Passwort entsprechen; gib am Ende das korrekte Passwort ein:“pass!”.

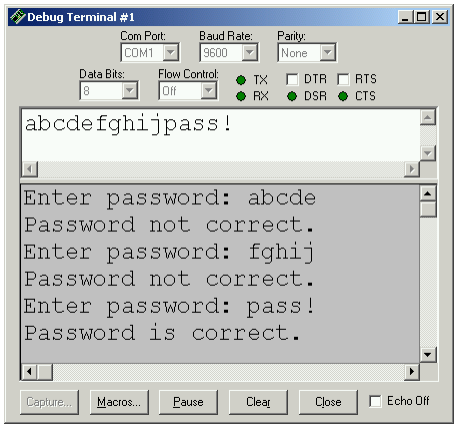


Abbildung 10-7
Passworteingabe im Fenster Senden

Klicke in das Fenster Senden (oben) und gib ein Passwort ein.

```
' What's a Microcontroller - PasswordChecker.bs2
' Check password entered in Debug Terminal's Transmit Windowpane.

' {$STAMP BS2}
' {$PBASIC 2.5}

Password       DATA    "pass!"               ' Store "secret" password here.

index          VAR     Nib                   ' Index variable.
temp           VAR     Byte                  ' Stores single char.
userEntry      VAR     Byte(5)               ' Store user entered password.

DO

  DEBUG "Enter password: "                   ' User instructions.

  DEBUGIN STR userEntry \5                   ' Get user input password.

  FOR index = 0 TO 4                         ' Check array against DATA
    READ Password + index, temp              ' Get next password char
    IF temp <> userEntry(index) THEN EXIT    ' Compare to user input,
  NEXT                                       ' exit if not equal.

  IF index <> 5 THEN                         ' If exit, then index not equal
    DEBUG CR,"Password not correct.", CR     ' to 5 and pass is not correct.
  ENDIF

LOOP UNTIL index = 5                         ' Only get out of loop when
                                             ' index = 5.

DEBUG CR, "Password is correct;", CR,        ' Program can move on when
          "program can continue..."          ' password is correct.
END
```

Du bist dran – Passwort ändern

- √ Änder die Direktive **`Password DATA`** so ab, dass damit ein neues Passwort aus fünf Zeichen eingerichtet wird.
- √ Durch Änderung von fünf verschiedenen Werten im Programm lässt es sich so verändern, dass nur noch ein Passwort aus vier Zeichen akzeptiert wird.

Änderung der Passwortabfrage

Die Aufgabe ist, das Programm PasswordChecker.bs2 in das Beispielprogramm aus Übung 3 einzupassen. Zwei Dinge werden uns helfen. Erstens muss das Passwortprogramm zu einem Unterprogramm so umgebaut werden, dass viele der Aufgaben des Originalprogramms übernommen werden. Zweitens, sollten die unterschiedlichen Programmteile durch kommentierte Zeilen markiert werden und dadurch mithelfen, die Arbeitsweise der beiden Programmteile transparenter machen.

Programmbeispiel: ReusablePasswordChecker.bs2

ReusablePasswordChecker.bs2 arbeitet ähnlich wie PasswordChecker.bs2, mit dem Unterschied, dass die Aufgabe an ein Unterprogramm übertragen und das Gesamtprogramm durch beschriftete Abschnitte strukturiert wurde.

- √ Überzeuge dich, dass das Programm genau so arbeitet wie PasswordChecker.bs2.
- √ Schau dir genau an, wie der **`DO…LOOP UNTIL`** Programmblock im Unterprogramm umgesetzt wurde.

```
' -----[ Title ]-------------------------------------------------------------
' What's a Microcontroller - ReusablePasswordChecker.bs2
' Check password entered in Debug Terminal's Transmit Windowpane.

' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ DATA Directives ]-----------------------------------------------

Password       DATA    "pass!"               ' Store "secret" password here.

' -----[ Variable Declarations ]-----------------------------------------

index          VAR     Nib                   ' Index variable.
temp           VAR     Byte                  ' Stores single char.
userEntry      VAR     Byte(5)               ' Store user entered password.
```

```
' -----[ Initialization Routine ]----------------------------------------

GOSUB Check_Password

' -----[ Main Routine ]---------------------------------------------------

' There is no main routine in this program.

DEBUG CR, "All Done"

END

' -----[ Subroutine - Check for Correct Password ]------------------------

Check_Password:

  DO

    DEBUG "Enter password: "                 ' User instructions.

    DEBUGIN STR userEntry \5                 ' Get user input password.

    FOR index = 0 TO 4                       ' Check array against DATA
      READ Password + index, temp            ' Get next password char
      IF temp <> userEntry(index) THEN EXIT  ' Compare to user input,
    NEXT                                     ' exit if not equal.

    IF index <> 5 THEN                       ' If exit, then index not equal
      DEBUG CR,"Password not correct.", CR   ' to 5 and pass is not correct.
    ENDIF

  LOOP UNTIL index = 5                       ' Only get out of loop when
                                             ' index = 5.

  DEBUG CR, "Password is correct."           ' Program can move on when
                                             ' password is correct.

RETURN                                       ' Return when pass is correct.
```

Für Fortgeschrittene: Du bist dran – Zwei Programme zusammenfügen

Wir haben die beiden folgenden Programme ausgetestet und auf Fehlerfreiheit überprüft: TerminalOperatedSensorArray.bs2 und ReusablePasswordChecker.bs2. Unsere nächste Aufgabe besteht darin, beide Programme zu einem zusammenzufügen, das die Passworteingabe überprüft und bei korrekter Eingabe den Nutzer über das Debug Terminal um die gewünschte Sensorabfrage bittet.

- √ Öffne im BASIC Stamp Editor die Programme ReusablePasswordChecker.bs2 und TerminalOperatedSensorArray.bs2.
- √ Speicher das Programm TerminalOperatedSensorArray.bs2 unter PasswordedSensorTerminal.bs2 ab.

Während des Kopiervorgangs einzelner Programmteile muss zwischen den beiden Programmen ReusablePasswordChecker.bs2 und PasswordedSensorTerminal.bs2 hin- und hergesprungen werden.

- √ Kopiere die Direktive **`Password DATA`** (inklusive der kommentierten Kopfzeilen mit der gestrichelten Linie) aus ReusablePasswordChecker.bs2 unmittelbar vor den **`I/O Definitions`** Abschnitt in PasswordedSensorTerminal.bs2.
- √ Kopiere die Variablendeklarationen aus ReusablePasswordChecker.bs2 und füge sie in PasswordedSensorTerminal.bs2 ein. Die Variablendeklarationen werden zu denen in PasswordedSensorTerminal.bs2 hinzugefügt, deshalb kopiere auch die kommentierten **`Variables`** Kopfzeilen.
- √ Kopiere den Abschnitt **`Initialization`** (mit den kommentierten Kopfzeilen) aus ReusablePasswordChecker.bs2 und füge ihn ein zwischen dem **`Variables`** Abschnitt und der **`Main Routine`** in PasswordedSensorTerminal.bs2.
- √ Kopiere den kompletten Abschnitt **`Subroutine`** aus ReusablePasswordChecker.bs2 und füge ihn nach dem letzten Unterprogramm in PasswordedSensorTerminal.bs2 ein.
- √ Teste PasswordedSensorTerminal.bs2 und überprüfe, ob es fehlerfrei arbeitet; wenn nicht, beseitige die Fehler.

ZUSAMMENFASSUNG

In diesem Kapitel haben wir erfahren, wie man einzelne Schaltungen austestet, bevor man sie in ein größeres System einbaut. Wir haben gelernt, wie mit dem Befehl **ON…GOSUB** ein Menüsystem aufgebaut wird und die Direktive **PIN** hat gezeigt, wie man I/O Pins mit Namen belegen kann. Wir wissen, wie der BASIC Stamp Editor entscheidet, wann ein Wert über einen Eingang eingelesen oder über einen Ausgang ausgegeben werden soll. Ein Passwortprogramm machte uns mit Variablenfeldern, dem Befehl **EXIT** und dem Formatierungsbefehl **STR** des Befehls **DEBUGIN** bekannt. Am Ende erhöhten wir die Funktionalität des Sensorenprogramms durch den Einbau eines Passwortprogramms.

Verständnisfragen

1. Wann und warum sollten Unterprogramme einzeln getestet werden, bevor man sie in einem Programm zusammenfasst?
2. Wie viele Programme aus anderen Kapiteln wurden von dir in diesem Kapitel benutzt?
3. Worin unterscheidet sich die Direktive **PIN** von den Direktiven **CON** und **VAR**?
4. Worin besteht der Unterschied zwischen **EXIT** und **END**?

Übungen

1. Beschreibe den 3-Stufen Prozess, mit dem die Schaltung *Piezoschallwandler* dem Projekt hinzugefügt wurde.
2. Änder das Programm PasswordChecker.bs2 von Seite 313 so ab, dass die Mitteilung "you entered: " zusammen mit dem Passwort auf dem Debug Terminal erscheint.

Projekte

1. Füge eine Schaltung *Piezoschallwandler* zum Sensorfeld, das in diesem Kapitel entwickelt wurde, hinzu und schreibe ein Programm, mit dem die Sensoren getestet werden können.
2. Veränder TerminalOperatedSensorArray.bs2 so, dass ein fünfter Menüeintrag erscheint, über den ein Schallwandler so angesprochen wird, dass er für 1,5 Sekunden einen Ton von 2 kHz abgibt.
3. Während du an den Übungen und Projekten zu diesem Buch gearbeitet hast, kam dir vielleicht die Idee: “Hey, eigentlich könnte ich es dazu nutzen, um >> trage

hier deine Projektidee ein: _____ <<!" Entwickle mit allem, was du bisher gelernt hast, dein eigenes technisches Gerät.

Lösungen

Q1. Wann: immer! Es ist leichter Fehler/Probleme zu finden und zu vermeiden.

Q2. Zwei. ReadPushbuttonState.bs2 aus Kapitel 3 und ReadPotWithRcTime.bs2 aus Kapitel 5.

Q3. Während die Direktive **CON** einer Zahl einen Namen zuweist und die Direktive **VAR** den Speicherzellen (RAM) einen Namen zuweist, weist die Direktive **PIN** den BASIC Stamp I/O Pins einen Namen zu. Dieser Name kann im Programm an Stelle der I/O Pinzahl verwendet werden. PBASIC bestimmt, ob der Pin als Ausgang (zum Senden von HIGH/LOW Signalen) oder als Eingang (zur Feststellung des Zustandes am I/O Pin) benutzt wird.

Q4. **END** beendet das Programm. Im Gegensatz dazu beendet **EXIT** eine Schleife und setzt das Programm in der der Schleife folgenden Programmzeile fort.

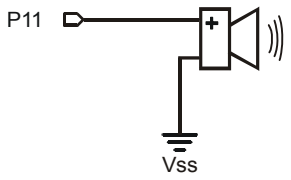
E1. a. Baue die Schaltung Piezoschallwandler auf.
b. Führe TestPiezoWithFreqout.bs2 aus Kapitel 8 aus und teste den Lautsprecher.
c. Füge den Piezoschallwandler in das Programm ein.

E2. Benutze zur Lösung die Formatierung **STR** im Befehl **DEBUG**. Im Text wurde die Formatierung **STR** mit dem Befehl **DEBUGIN** eingeführt, so dass man durchaus annehmen kann, dass **STR** auch mit **DEBUG** zusammenarbeitet. Zur Lösung füge die folgende Programmzeile unterhalb der **DEBUGIN** Anweisung hinzu.

```
DEBUG CR, CR, "You entered: ", STR userEntry, CR
```

P1. Projekt 1 benutzt die Schaltung aus Abbildung 10-1. S. 302, der ein Schallwandler an P11 hinzugefügt wurde.

```
' What's a Microcontroller –
' TestPiezoWithFreqout.bs2
' Send a tone to the piezo speaker
' using the FREQOUT command.
'{$STAMP BS2}
'{$PBASIC 2.5}

DEBUG "Tone sending...", CR
FREQOUT 11, 1500, 2000
DEBUG "Tone done."
```



P2. Projekt 2 nutzt die Schaltung von Projekt 1.

```
' What's a Microcontroller - Ch10Prj02_AddSpeakerToSensorArray.bs2
' Use Debug Terminal to choose to read one of five sensors.
' Project 2 - Add a 5th menu item that makes a piezo speaker beep
' for 1.5 seconds at a frequency of 2kHz.
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ I/O Definitions ]---------------------------------------------

Pb1Pin          CON     3
Pb2Pin          CON     4
PotPin          CON     7
PhotoPin        CON     9
SpkrPin         CON     11

' -----[ Constants ]-------------------------------------------------

DelayRc         CON     100
DelayReader     CON     1500

' -----[ Variables ]-------------------------------------------------

request         VAR     Nib
time            VAR     Word

' -----[ Main Routine ]----------------------------------------------

DO
  GOSUB Display_Menu
  GOSUB Get_Request
  ON request GOSUB Read_Pushbutton_1, Read_Pushbutton_2,
                   Read_Pot, Read_Photoresistor, Beep_Speaker
LOOP

' -----[ Subroutines]------------------------------------------------

Display_Menu:

  DEBUG CLS, "MENU: ", CR,CR,
             "1) Pushbutton 1", CR,
             "2) Pushbutton 2", CR,
             "3) Potentiometer", CR,
             "4) Photoresistor", CR,
             "5) Piezo Speaker", CR, CR
  RETURN

Get_Request:
  DEBUGIN DEC1 request
  request = request - 1
  RETURN
```

```
Read_Pushbutton_1:
  DEBUG CLS, ? IN3
  PAUSE DelayReader
  RETURN

Read_Pushbutton_2:
  DEBUG CLS, ? IN4
  PAUSE DelayReader
  RETURN

Read_Pot:
  HIGH 7
  PAUSE DelayRc
  RCTIME 7, 1, time
  DEBUG CLS, DEC time, "     "
  PAUSE DelayReader
  RETURN

Read_Photoresistor:
  HIGH 9
  PAUSE DelayRc
  RCTIME 9, 1, time
  DEBUG CLS, DEC time, "     "
  PAUSE DelayReader
  RETURN

Beep_Speaker:
  DEBUG CLS, "2000 Hz beep for 1.5 seconds", CR
  FREQOUT SpkrPin, 1500, 2000                ' 2000 Hz beep for 1.5 s
  RETURN
```


P3. Wenn du eine interessante Anwendung mit dem *What's a Microcontroller?* Bausatz entwickelt hast und dich mit anderen darüber austauschen möchtest, kannst du dies tun über die Stamps In Class Yahoo Group. Weitere Hinweise findest du auf der Web Site unter www.stampsinclass.com.

Weiterführende Bücher

Im Vorwort findet sich ein Hinweis auf alle Bücher aus dem Stamps in Class Lehrgang. Sie sind kostenlos über das Internet unter der Adresse www.parallax.com herunterzuladen und befinden sich auch auf der Parallax CD. Es ist ein enormer Wissenspool in dem sich sehr viele Anregungen für eigene Untersuchungen aus den Bereichen Elektronik, Programmierung, Robotics und Ingenieurwissenschaften finden. Wir empfehlen Praktikern, Lehrkräften, Studenten, Hobbyisten und neugierig Gebliebenen unbedingt die Site www.parallax.com zu besuchen.

A

Anhang A: USB-Seriell Adapter

Wir empfehlen für die Parallax Produkte den US232B/LC USB-Seriell Adapter der Firma Future Technology Devices International. Das US232B/LC Paket besteht aus einem Adapterkabel, wie in Abbildung A-1 dargestellt, und einer Mini-CD ROM, mit Treibern für Microsoft Windows und weitere Betriebssysteme.

Abbildung A-1
FTDI's
US232B/LC USB
- Seriell Adapter

Parallax
Bestellnummer
800-00030.

US232B/LC Treibersoftware herunterladen: Softwaretreiber und weitere Informationen über das Produkt gibt es unter: http://www.ftdichip.com/FT232.htm.

Die Fa. Konrad Elektronik bietet unter der Bestellnummer 98 24 17-88 ein USB-Seriell Adapterkabel mit Treibersoftware an, das problemlos mit den Parallax Boards zusammenarbeitet.

Anhang B: Ausstattung und Bauteile

Der Umfang der Bauteile kann sich verändern: Die in diesem Anhang angegebenen Bauteilmengen und die Materiallisten sind ständigen Veränderungen unterworfen. Bei Fragen zu einem speziellen Bauteil oder Mengenangaben, kontaktieren Sie bitte Parallax unter www.parallax.com → Company → Contact Parallax, oder senden Sie eine email an stampsinclass@parallax.com.

Für die Übungen dieses Buches wird einer der beiden folgenden Parallax Bausätze benötigt:

Option 1:

- **Board of Education® Full Kit (#28102)** - *UND-*
- **What's a Microcontroller Parts Kit (#28152 mit Buch, #28122 ohne Buch)**

Jeder Bausatz kann einzeln erworben werden. Das Board of Education Full Kit (Inhalt siehe unten) ist der empfohlene Basis-Bausatz für alle Stamps in Class Lehrgänge.

Board of Education® Full Kit (#28102)
Bauteile und Mengenangaben (ohne Gewähr)

Parallax Part #	Description	Quantity
550-00022	Board of Education	1
800-00016	Steckverbindungen	10
BS2-IC	BASIC Stamp® 2 Modul	1
800-00003	Serielles Kabel	1
750-00008	DC Netzteil – 9 V, 300 mA	1
27000	Parallax CD – Softwaresammlung	1
700-00037	Gummifüße – 4 Stück	1

Das What's a Microcontroller Parts Kit kann separat (#28122) oder zusammen mit dem Buch *What's a Microcontroller?* (#28152) erworben werden. Mit diesem Bauteilesatz lassen sich die jeweiligen Übungen und Projekte des aktuell vorliegenden Lehrgangs durchführen. Der Bauteilesatz von *What's a Microcontroller* ist in einer Tabelle auf der folgenden Seite aufgelistet.

What's a Microcontroller Parts Kit #28122 **What's a Microcontroller Parts & Text #28152** Bauteile und Mengenangaben (ohne Gewähr)		
Parallax Teile #	**Beschreibung**	**Menge**
150-01020	Widerstand, 5%, 1/4W, 1 kΩ	10
150-01030	Widerstand, 5%, 1/4W, 10 kΩ	4
150-01040	Widerstand, 5%, 1/4W, 100 kΩ	2
150-02020	Widerstand, 5%, 1/4W, 2 kΩ	2
150-02210	Widerstand, 5%, 1/4W, 220 Ω	6
150-04710	Widerstand, 5%, 1/4W, 470 Ω	6
152-01031	Potentiometer - 10 kΩ	1
200-01031	Kondensator, 0.01 µF, 50 V	1
200-01040	Kondensator, 0.1 µF, 100 V	2
201-01080	Kondensator, 1000 µF, 10 V	1
201-03080	Kondensator 3300 µF, 16 V	1
28123	*What's a Microcontroller?* Buch (Nur unter #28152 erhältlich.)	1
350-00001	LED - Grün - T1 3/4	2
350-00005	LED – DUO - T1 3/4	1
350-00006	LED – Rot - T1 3/4	2
350-00007	LED – Gelb - T1 3/4	2
350-00009	Fotowiderstand	1
350-00027	7-Segment LED Anzeige	1
400-00002	Taster – als Schließer	2
451-00303	3 Pin Stiftleiste	1
500-00001	Transistor – 2N3904	1
604-00010	10 kΩ Digitalpotentiometer	1
800-00016	3" Jumper – PE 10 Stück	2
900-00001	Piezo Schallwandler	1
900-00005	Parallax Standard Servo	1

Option 2:

- **BASIC Stamp What's a Microcontroller Kit (#90005)**

Dieses Paket besteht aus dem Buch, den Bauteilen des *What's a Microcontroller* Lehrgangs, dem HomeWork Board und zusätzlichen Bauteilen. An Stelle des Board of Education mit separatem BASIC Stamp 2 Modul kann auch das BASIC Stamp HomeWork Board mit dem Buch *What's a Microcontroller?* durchgearbeitet werden. Das HomeWork Board lässt sich, bis auf geringe Einschränkungen, ebenfalls in den meisten Übungen der Stamps in Class Lehrgänge unmittelbar einsetzen. Folgende Teile sind im Bausatz *BASIC Stamp What's a Microcontroller Kit* enthalten:

BASIC Stamp® What's a Microcontroller Kit (#90005) Bauteile und Mengenangaben (ohne Gewähr)		
Parallax Teile #	**Beschreibung**	**Menge**
555-28158	HomeWork Board™ mit Steckbrett	1
28123	What's a Microcontroller Buch	1
27000	Parallax CD – Softwaresammlung	1
800-00003	Serielles Kabel	1
28122	What's a Microcontroller Bauteile	1
700-00037	Gummifüße – 4 Stück	1

Hinweis an Lehrkräfte: Das HomeWork Board ist für Schulen kostengünstiger in Packungsgröße von je 10 Einheiten erhältlich. Es liegt preislich deutlich unter dem *Board of Education + BASIC Stamp 2 Modul*. Bitte wenden Sie sich an den entsprechenden Vertriebshändler und erfragen Sie die aktuellen Preise.

BASIC Stamp® HomeWork Board™ Ten-Pack (#28158)		
Parallax Teile #	**Beschreibung**	**Menge**
28158	BASIC Stamp® HomeWork Board™ (BASIC Stamp 2 auf Board integriert)	10

Anhang C: Komponenten des BASIC Stamp Moduls und ihre Funktionen

C

Die BASIC STAMP® 2

Abbildung C-1 zeigt ein BASIC Stamp® 2 Mikrocontroller Modul von Parallax Inc.. Die einzelnen Komponenten mit ihren Funktionen sind jeweils beschrieben .

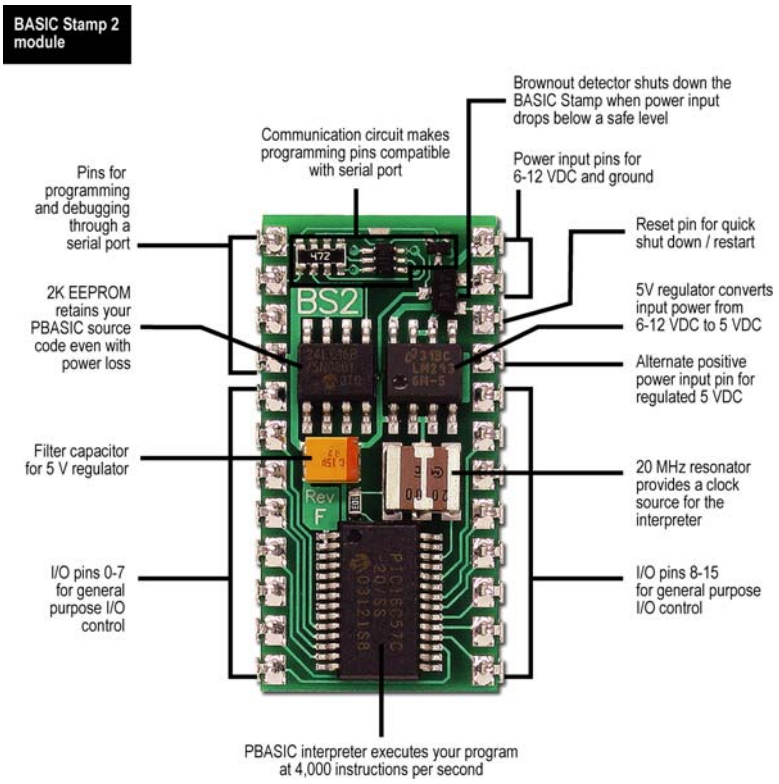

BASIC Stamp 2 module
Brownout detector shuts down the BASIC Stamp when power input drops below a safe level
Communication circuit makes programming pins compatible with serial port
Power input pins for 6-12 VDC and ground
Pins for programming and debugging through a serial port
Reset pin for quick shut down / restart
2K EEPROM retains your PBASIC source code even with power loss
5V regulator converts input power from 6-12 VDC to 5 VDC
Alternate positive power input pin for regulated 5 VDC
Filter capacitor for 5 V regulator
20 MHz resonator provides a clock source for the interpreter
I/O pins 0-7 for general purpose I/O control
I/O pins 8-15 for general purpose I/O control
PBASIC interpreter executes your program at 4,000 instructions per second

Abbildung C-1: Komponenten und ihre Funktionen eines BASIC Stamp® 2 Mikrocontroller Moduls von Parallax Inc.

Das Board of Education® Rev C

Das Board of Education® Rev C von Parallax Inc. mit seinen Komponenten und Funktionen zeigt Abbildung C-2.

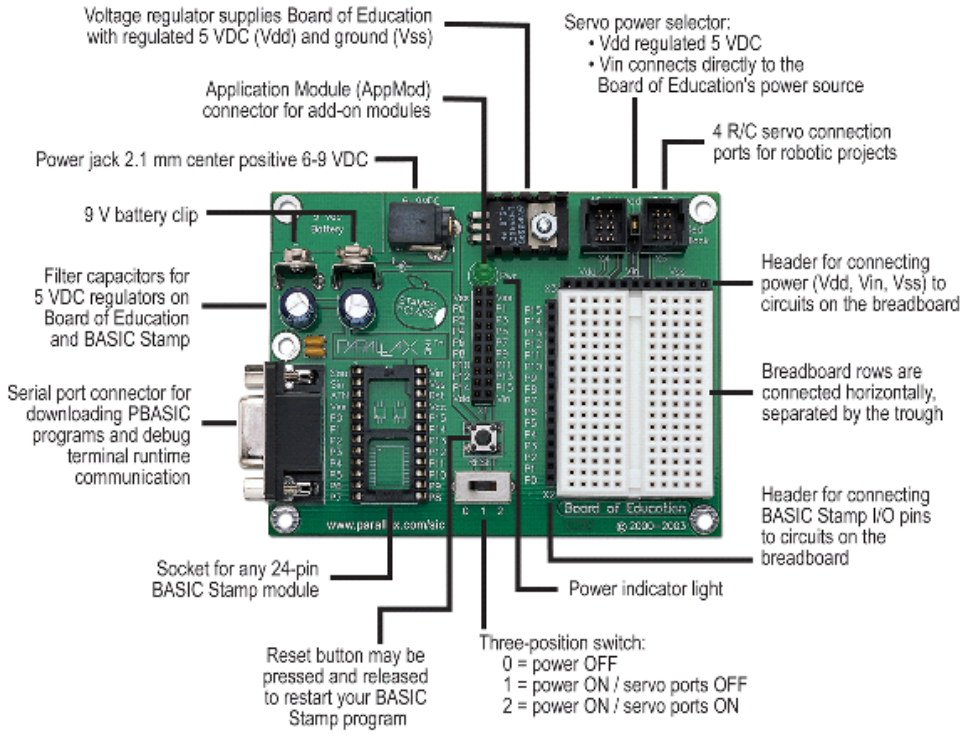

Voltage regulator supplies Board of Education with regulated 5 VDC (Vdd) and ground (Vss)
Servo power selector:
• Vdd regulated 5 VDC
• Vin connects directly to the Board of Education's power source
Application Module (AppMod) connector for add-on modules
4 R/C servo connection ports for robotic projects
Power jack 2.1 mm center positive 6-9 VDC
9 V battery clip
Header for connecting power (Vdd, Vin, Vss) to circuits on the breadboard
Filter capacitors for 5 VDC regulators on Board of Education and BASIC Stamp
Breadboard rows are connected horizontally, separated by the trough
Serial port connector for downloading PBASIC programs and debug terminal runtime communication
Header for connecting BASIC Stamp I/O pins to circuits on the breadboard
Socket for any 24-pin BASIC Stamp module
Power indicator light
Reset button may be pressed and released to restart your BASIC Stamp program
Three-position switch:
0 = power OFF
1 = power ON / servo ports OFF
2 = power ON / servo ports ON

Abbildung C-2: Board of Education® Rev C von Parallax. Inc. für BASIC Stamp® Microcontroller Module

Das BASIC Stamp® HomeWork Board™

Das BASIC Stamp® HomeWork Board™ von Parallax Inc. mit seinen Komponenten und Funktionen zeigt Abbildung C-3.

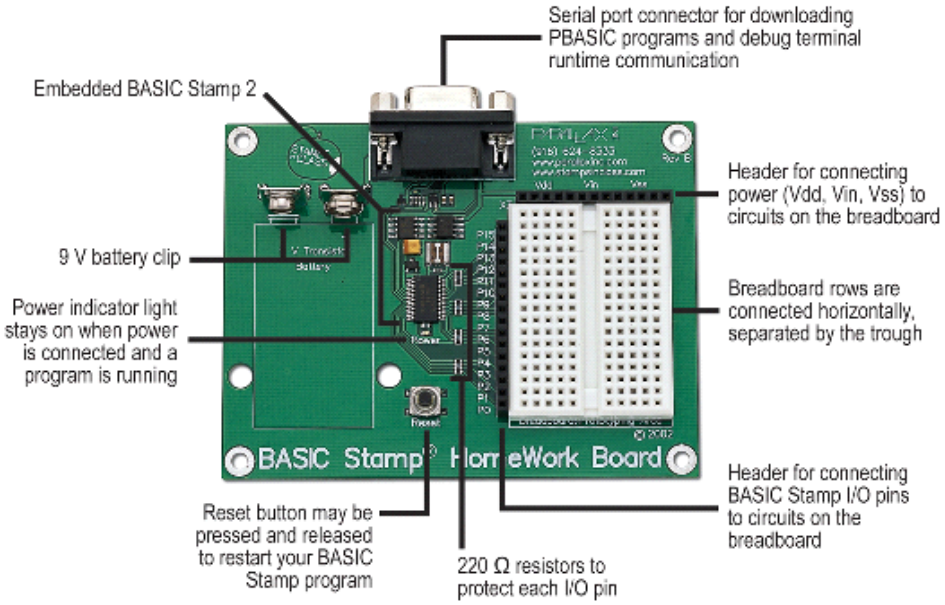

Serial port connector for downloading PBASIC programs and debug terminal runtime communication
Embedded BASIC Stamp 2
Header for connecting power (Vdd, Vin, Vss) to circuits on the breadboard
9 V battery clip
Breadboard rows are connected horizontally, separated by the trough
Power indicator light stays on when power is connected and a program is running
Header for connecting BASIC Stamp I/O pins to circuits on the breadboard
Reset button may be pressed and released to restart your BASIC Stamp program
220 Ω resistors to protect each I/O pin

Abbildung C-3: BASIC Stamp® HomeWork Board™ von Parallax Inc. mit integriertem BASIC Stamp® 2 Mikrocontroller Modul (SMD).

Das Board of Education® Rev B

Abbildung C-4 zeigt das Board of Education® Rev B von Parallax, Inc. mit seinen Komponenten und Funktionen.

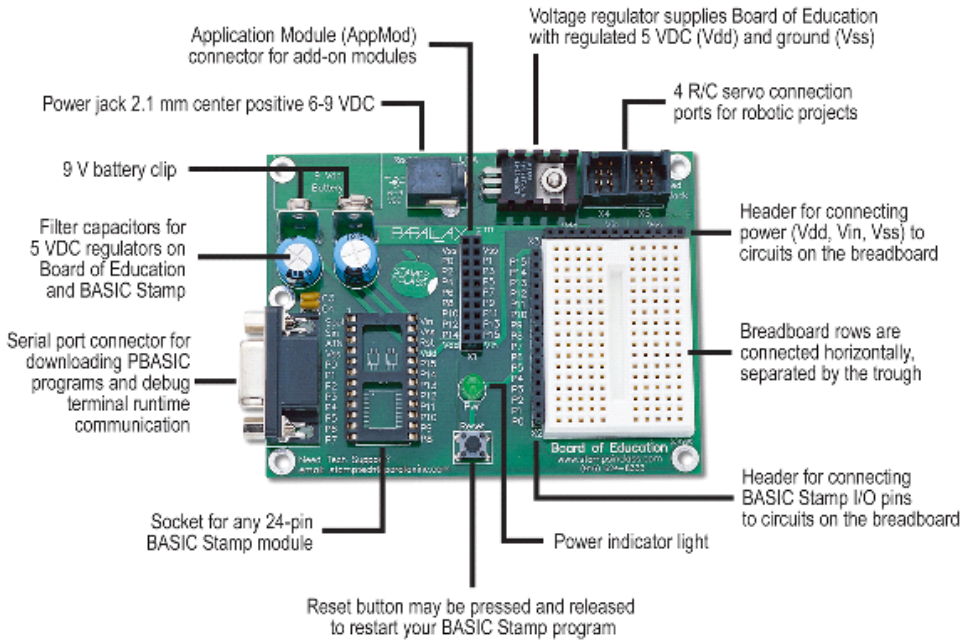

Application Module (AppMod) connector for add-on modules
Voltage regulator supplies Board of Education with regulated 5 VDC (Vdd) and ground (Vss)
Power jack 2.1 mm center positive 6-9 VDC
4 R/C servo connection ports for robotic projects
9 V battery clip
Filter capacitors for 5 VDC regulators on Board of Education and BASIC Stamp
Header for connecting power (Vdd, Vin, Vss) to circuits on the breadboard
Serial port connector for downloading PBASIC programs and debug terminal runtime communication
Breadboard rows are connected horizontally, separated by the trough
Header for connecting BASIC Stamp I/O pins to circuits on the breadboard
Socket for any 24-pin BASIC Stamp module
Power indicator light
Reset button may be pressed and released to restart your BASIC Stamp program

Abbildung C-4: Board of Education® Rev B von Parallax. Inc. für BASIC Stamp® Module

Anhang D: Batterien und Netzteile

9 V BATTERIEN

Beste Ergebnisse erzielt man mit 9 V Batterien.

9 V Batterie Spezifikationen: Wähle Batterien mit ähnlichen Merkmalen:

Nicht wiederaufladbar	**Wiederaufladbare Akkus**
• Alkaline	• Ni-Cad (Nickel Cadmium) • Ni-MH (Nickel Metal Hydride)

Beste Ergebnisse erzielt man mit Batterien von mindestens 100 mAh.

Nicht alle Ladegeräte können Ni-MH und Ni-Cad Batterien aufladen. Das Ladegerät muss für den eingesetzten Akku-Batterietyp unbedingt vorgesehen sein (Ni-Cad und/oder Ni-MH), um unvorhersehbare Schäden zu vermeiden.

NETZTEILE

Es wird empfohlen, beim Kauf eines Bausatzes ein entsprechendes Netzteil gleich mit zu bestellen. Bei allen Übungen in diesem Buch, die den Servo einsetzen, muss der Jumper zwischen den Wannensteckern X4 und X5 auf Vdd gesetzt werden. Die in Table D-1 aufgeführten Angaben sind Richtwerte. Steckernetzteile müssen mit einem 2,1 mm Hohlstecker (außen -, innen +) ausgerüstet sein.

Table D-1: Kenngrößen für Netzteile

Polarität/Stecker Spezifikation	**Eingang**		**Ausgang**	
	Volt	**Hz**	**VDC**	**mA**
Stecker-Schaltnetzteil 2,1 mm Hohlstecker, Polarität: Außen -, Innen +	240	50	9	800-1000 (empfohlen)

DIE GLEICHSPANNUNGSVERSORGUNG

Es sollten mit dem BASIC Stamp HomeWork Board oder dem Board of Education Steckernetzteile mit den folgenden Leistungsangaben eingesetzt werden:

Eingangsdaten

Für die EU gilt: 240 V, 50 Hz. Für die USA und Kanada gilt: 120 VAC, 60 Hz.

Ausgangsdaten

6 Volt, 800 mA
Die Ausgangsstromstärke kann auch geringfügig höher liegen.

Steckverbinder

Das Board of Education verfügt über eine DC-Kupplung zum Anschluss eines Steckernetzteiles und einen 9V Batterieclip, das HomeWork Board verfügt nur über einen 9V Batterieclip.

Steckernetzteil

Abbildung D-1 zeigt ein Gleichspannungs-Steckernetzteil (US-Version), wie es im allgemeinen mit einer BASIC Stamp und dem Board of Education benutzt werden kann. Es verfügt über einen 2.1 mm Hohlsteckeradapter mit positivem Mittelstecker.

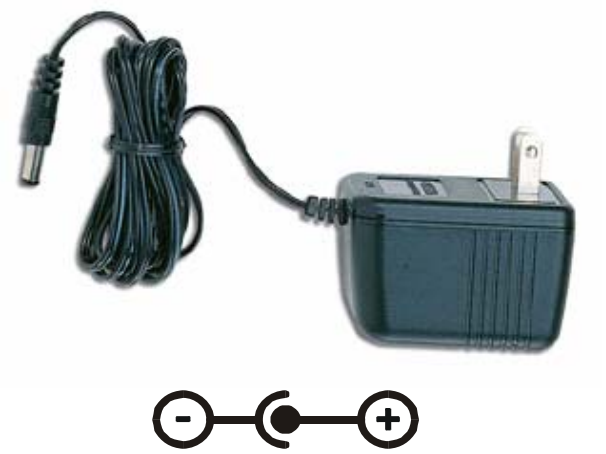


Abbildung D-1
Gleichspannungsversorgung mit einem Steckernetzteil dessen Mittelelektrode positiv belegt ist (US-Version)

9 V Batterieclip

Abbildung D-2 zeigt ein Steckernetzteil mit einem passenden 9 V Batterieclip, der mit dem BASIC Stamp HomeWork Board verbunden werden kann. Beachte den nachfolgenden HINWEIS.

Abbildung D-2
Steckernetzteil mit 9 V Batterieclip (US-Version)

HINWEIS – Vorsicht bei Universal-Steckernetzteilen mit umkehrbarer Polarität

Abbildung D-3 zeigt einen gängigen Fehler im Umgang mit Universalnetzteilen mit wählbarer Polarität. Bei falsch gewählter Polarität nehmen weder die BASIC Stamp, noch das Board of Education oder Homework Board Schaden, zerstört wird aber der mit Vin verbundene Parallax Standard Servo. Verpolungsschutz für einen Servo bietet nur das Board of Education Rev C (Jumper auf Vdd).

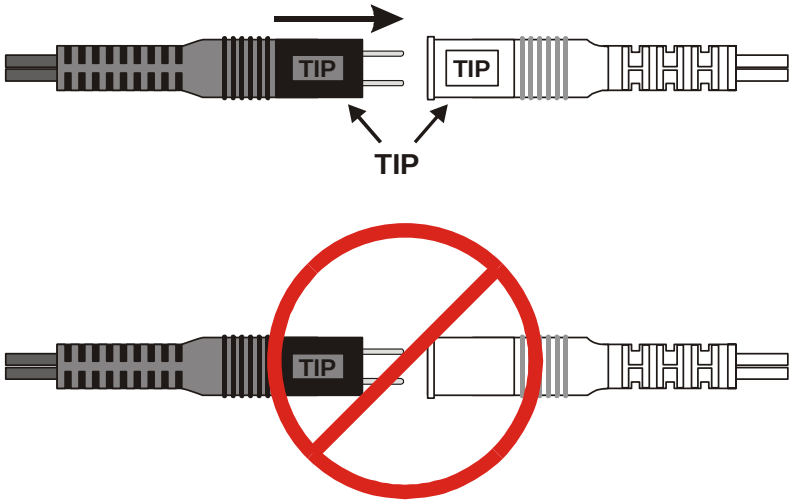


Abbildung D-3
Polarität an einem Universalstecker-netzteil

Vorsicht mit sog. „Ersatzbatterien“

Viele Ersatzbatterien sind nur für geringe Leistungen ausgelegt. Ihre Stromstärken, die im Bereich von ca. 10 mA liegen, sind für viele Übungen dieses Buches nicht ausreichend dimensioniert. Zwei LEDs mit einem Schutzwiderstand von 220 Ω ziehen einen Strom von 14.5 mA, die BASIC Stamp zusätzlich zwischen 3 und 7 mA. Ein Servo benötigt um die 100 mA. Dies zeigt, dass sog. “Ersatzbatterien“ keine Alternative sind.

HINWEIS: Die Power LED auf dem Board of Education (oder HomeWork Board) ist ein guter Indikator bei Überlastung, da sie zu flackern beginnt oder verlischt, sobald mehr Strom benötigt wird, als zur Verfügung gestellt wird.

Anhang E: Fehlersuche

Hier sind einige Fehlerhinweise und Lösungsschritte aufgelistet, die beim Arbeiten mit der BASIC Stamp und dem BASIC Stamp Editor auftreten können:

- √ Bei einem Board of Education Rev C muss der Schiebeschalter in Position 1 gebracht werden..
- √ Tausche schwache Batterien oder defekte Netzteile durch eine neue 9 V Batterie aus.
- √ Achte darauf, dass das serielle Kabel sowohl mit dem COM-Port des Rechners als auch mit der DB9-Buchse des Board of Education oder dem BASIC Stamp HomeWork Board fest verbunden ist.
- √ Überprüfe durch Sichtnahme, ob ein einfaches serielles und kein NULL-MODEM-KABEL verwendet wird. Die meisten Null-Modem-Kabel sind mit *NULL* oder *Null Modem* gekennzeichnet. Wenn das der Fall ist, tausche das Kabel sofort aus.
- √ Stelle sicher, dass auf dem Computer keine Palmtop-Kommunikationssoftware im Hintergrund läuft.

Wird eine BASIC Stamp mit einem Board of Education eingesetzt, prüfe auch die folgenden Punkte:

- √ Überprüfe, ob die BASIC Stamp seitenrichtig in die Fassung eingesetzt wurde und die Referenzkerbe in die korrekte Richtung zeigt. Siehe Abbildung 1-28 auf Seite 18.
- √ Wird ein Steckernetzteil benutzt, muss es sowohl mit der Steckdose als auch mit der Buchse am Board of Education verbunden sein. Überprüfe, ob die grüne LED des Board of Education aufleuchtet, wenn das Steckernetzteil angeschlossen ist.
- √ Überprüfe, ob die BASIC Stamp fest in der Fassung steckt. Achte besonders darauf, dass alle Pins der BASIC Stamp in den jeweiligen Buchsen der Fassung stecken und nicht umgeknickt wurden. Schalte die Spannungsversorgung ab und drücke mit dem Daumen das Modul vorsichtig in die Fassung.

Zeigt das Identification Fenster eine ähnliche Meldung wie in Abbildung E-1, bedeutet dies, dass der BASIC Stamp Editor die BASIC Stamp an der seriellen Schnittstelle nicht

erkannt hat. In den Spalten *Loopback* und *Echo* wird jeweils "No" angezeigt. Versuche wie folgt vorzugehen:

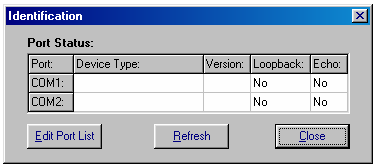


Abbildung E-1
Identification Fenster

Beispiel: BASIC Stamp 2 an den COM Ports nicht identifiziert

- √ Schließe das Fenster *Identification*.
- √ Stelle sicher, dass das serielle Kabel fest verbunden ist.
- √ Teste erneut über *Run* → *Identify*.
- √ Ist der gewählte COM-Port bekannt, erscheint aber nicht im Fenster *Identification*, füge ihn über *Edit Port List* hinzu und teste erneut mit *Run* → *Identify*.
- √ Verfügt der Rechner über mehrere COM-Ports, verbinde das Board of Education oder das BASIC Stamp HomeWork Board mit einem anderen Port und teste wieder über *Run* → *Identify*.
- √ Als letzte Möglichkeit sollte, wenn weitere Computer zur Verfügung stehen, die Verbindung mit einem anderen Computer überprüft werden.

Wird die Fehlermeldung "No BASIC Stamp Found" angezeigt, während der Test *Run* → *Identify* in beiden Spalten "Yes" für einen bestimmten COM Port ausgibt, sollten die Einstellungen der FIFO Buffer abgeändert werden. Dies kann gelegentlich bei den Betriebssystemen Microsoft Windows® 98 und XP auftreten. Notiere die COM-Ports mit dem „Yes" Eintrag und gehe wie folgt vor:

<u>Windows® 98:</u>

- √ Klicke auf *Start*.
- √ Wähle *Settings*→ *Control Panel* → *System* → *Device Manager* → *Ports (COM & LPT)*.
- √ Wähle den COM Port, der über *Run* → *Identify* erkannt wurde.
- √ Wähle *Properties* → *Port Settings* → *Advanced*.
- √ Lösche die Markierung im Feld "Use FIFO Buffers" und klicke auf *OK*.
- √ Wähle *OK*, schließe alle Fenster und kehre zum BASIC Stamp Editor zurück.
- √ Versuche erneut, ein Programm herunterzuladen.

Windows® 2000:

- √ Klicke auf *Start*.
- √ Wähle *Settings → Control Panel → System → Hardware → Device Manager → Ports (COM & LPT)*.
- √ Wähle den COM Port, der über *Run → Identify* erkannt wurde.
- √ Wähle → *Port Settings → Advanced*.
- √ Lösche die Markierung im Feld "Use FIFO Buffers" und klicke auf *OK*.
- √ Wähle *OK*, schließe alle Fenster und kehre zum BASIC Stamp Editor zurück.
- √ Versuche erneut, ein Programm herunterzuladen.

Windows® XP:

- √ Klicke auf *Start*.
- √ Wähle *Control Panel → Printers and Other Hardware.*
- √ Wähle im *"*See Also*"* Fenster *System.*
- √ Wähle *Hardware → Device Manager → Ports.*
- √ Gib die COM-Port Zahl ein, die über *Run→ Identify* erkannt wurde.
- √ Wähle *Port Settings → Advanced.*
- √ Lösche die Markierung im Feld "Use FIFO Buffers" und klicke auf *OK*.
- √ Wähle *OK*, schließe alle Fenster und kehre zum BASIC Stamp Editor zurück.
- √ Versuche erneut, ein Programm herunterzuladen.

Windows® XP Pro:

- √ Klicke auf *Start*.
- √ Wähle *Control Panel → System → Hardware → Device Manager → Ports(COM & LPT1)*.
- √ Gib die COM-Port Zahl ein, die über *Run→ Identify* erkannt wurde.
- √ Wähle *Properties → Port Settings → Advanced.*
- √ Lösche die Markierung im Feld "Use FIFO Buffers" und klicke auf *OK*.
- √ Wähle *OK*, schließe alle Fenster und kehre zum BASIC Stamp Editor zurück.
- √ Versuche erneut, ein Programm herunterzuladen.

Führt keine der vorgestellten Lösungen zum Ziel, gehe im Internet auf die Site www.parallax.com und folge dem Support Hinweis oder sende eine email an support@parallax.com.

Anhang F: Grundlagen der Elektrizitätslehre

Was versteht man unter einem Elektron? Atome bestehen aus positiven Protonen, ungeladenen Neutronen und negativ geladenen Elektronen. Protonen und Neutronen bilden den Kern eines Atoms, der von Elektronen umgeben ist. Elektronen stoßen sich gegenseitig ab, Protonen und Elektronen ziehen sich gegenseitig an.

Welche Eigenschaften haben Ladungen? Es gibt negative und positive Ladungen. Gleichnamige Ladungen stoßen sich ab, ungleichnamige ziehen sich an. Ein Molekül, das über mehr Elektronen als Protonen verfügt, bezeichnet man als negativ geladen. Umgekehrt wird ein Molekül mit weniger Elektronen als Protonen als positiv geladen bezeichnet. Moleküle, die gleich viele Protonen und Elektronen haben, verhalten sich nach außen elektrisch neutral.

Was meint man mit dem Begriff Spannung? Die Kraft, mit der die Elektronen bewegt werden, ist von den Ladungen, die sich anziehen bzw. abstoßen, abhängig. Genauer gesagt, von dem Unterschied zwischen den Ladungsstärken (sog. Potentiale) zum Beispiel am Anfang und Ende eines Drahtes. Dabei bewegen sich die Elektronen vom Ort mit negativem Ladungsüberschuss (- Pol) zum Ort mit negativem Ladungsmangel (+ Pol). Der Spannungsbegriff lässt sich auch mit einer Kugel, die einen Abhang herunterrollen möchte, vergleichen. Die Steigung des Hangs ist ein Maß für die Spannung. Je größer die Steigung, desto höher die Spannung, desto schneller rollt die Kugel (Elektron). Die Höhe des Hangs ist ohne Bedeutung, da es nur auf den Höhenunterschied ankommt, den die Kugel durchläuft.

Die Einheit Volt ist ein Maß für die elektrische Spannung zwischen zwei Punkten. Sie wird mit dem Buchstaben V abgekürzt. Die Batterie des Board of Education oder HomeWork Board muss 9 Volt bereitstellen, Autos verfügen über 12 Volt Batterien und in Uhren, Fernbedienungen und Taschenrechnern finden sich AAA Batterien von je 1,5 Volt.

Was versteht man unter elektrischem Strom? Die Zahl der negativen Ladungen die sich pro Zeiteinheit durch einen bestimmten Leitungsquerschnitt bewegen, bezeichnet man als Strom. Die Stromstärke wird mit dem großen Buchstaben I bezeichnet.

Was versteht man unter Ampere? Das Ampere ist die Einheit für die elektrische Stromstärke; sie wird mit dem Buchstaben A abgekürzt. In den Schaltungen dieses Lehrgangs flossen nur sehr geringe Stromstärken, weit unterhalb von einem Ampere, während in anderen Geräten des täglichen Umgangs wesentlich höhere Stromstärken fließen (z.B. im Auto mehrere Ampere).

Was versteht man unter einem Widerstand? Wird die Bewegung freier Ladungsträger (Elektronen) in einem Stromkreis behindert, spricht man von einem Widerstand.

Die Einheit Ohm ist die Basiseinheit für den Widerstand. Sie wird mit dem griechischen Buchstaben Omega (Ω) abgekürzt.

Was ist ein Leiter? Kupfer und Silber setzen dem Strom nahezu keinen Widerstand entgegen. Sie sind gute Leiter im Gegensatz zu Gummi, Holz und Kunststoff, die als Nichtleiter oder Isolatoren bezeichnet werden.

BONUSÜBUNG: OHMSCHES GESETZ, SPANNUNG UND STROMSTÄRKE

In dieser Übung werden einige der obigen Definitionen angewendet.

Bauteile zum Ohmschen Gesetz

1 Widerstand – 220 Ω (rot-rot-braun)
1 Widerstand – 470 Ω (gelb-violett-braun)
1 Widerstand – 1 kΩ (braun-schwarz-rot)
1 Widerstand – 2 kΩ (rot-schwarz-rot)
1 LED – beliebige Farbe

Testschaltung

Der Widerstandswert von R_i in Abbildung F-1 kann ausgetauscht werden. Bei kleinerem Widerstand erhöht sich die Stromstärke durch die LED: sie leuchtet heller. Ein höherer Widerstand senkt die Stromstärke und die LED leuchtet nur noch schwach.

- √ Trenne das Board of Education oder HomeWork Board bei jeder Schaltungsänderung von der Spannungsquelle.
- √ Baue die Schaltung nach Vorlage auf; beginne mit einem 220 Ω Widerstand.
- √ Tausche den Widerstand 220 Ω gegen einen 470 Ω Widerstand aus. Wie reagiert die LED?
- √ Wiederhole den Vorgang mit einem 1 kΩ und anschließend mit einem 2 kΩ Widerstand und beobachte die Leuchtstärke der LED.

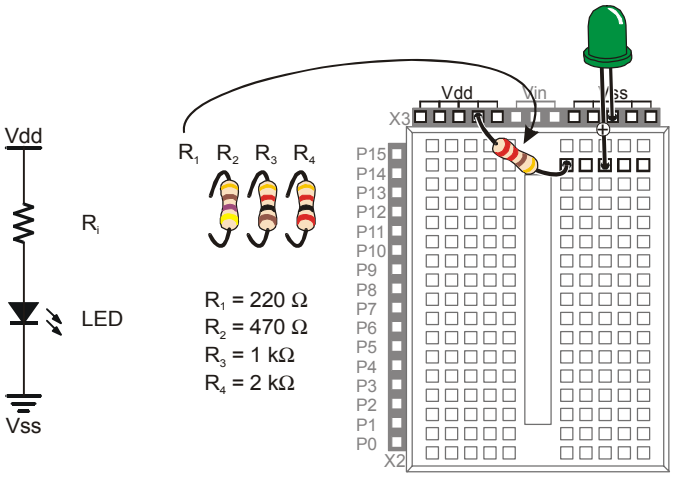


Abbildung F-1
LED Testschaltung

F

Arbeitest du mit einer 9 V Batterie, lassen sich auch hiermit Vergleiche anstellen. Vin ist direkt mit dem + Anschluss, Vss mit dem – Pol der Batterie verbunden. Vdd liefert eine stabilisierte Spannung von 5 V. Das entspricht knapp der Hälfte einer 9 V Batterie.

- √ Arbeitest du nicht mit einer 9 Volt Batterie, mache weiter mit dem Abschnitt: **Berechnung der Stromstärke**
- √ Beginne mit der Schaltung in Abbildung F-1 und starte mit einem 1 kΩ Widerstand.
- √ Notiere, wie hell die LED aufleuchtet.
- √ Trenne das Board von der Spannungsversorgung.
- √ Verbinde den Anschlussdraht des Widerstandes statt mit Vdd jetzt mit Vin.
- √ Lege die Spannungsversorgung wieder an und beobachte die Helligkeit der LED. Leuchtet sie heller?

BENUTZE AUF KEINEN FALL für die Vin Schaltung die Widerstandswerte 220 oder 470 Ω. Die Stromstärke durch die LED könnte die LED zerstören.

Berechnung der Stromstärke

Im BASIC Stamp Manual werden Hinweise gegeben, mit welcher Stromstärke das Modul belastet werden darf. Werden diese Grenzwerte nicht eingehalten, kann die

BASIC Stamp beschädigt werden. Wichtig ist, wie hoch die Stromstärke an einem I/O Pin und wie hoch sie insgesamt an allen I/O Pins sein darf.

Belastungsregeln für die BASIC Stamp I/O Pins

- Ein I/O Pin Ausgang kann mit 20 mA belastet werden. Wird ein `HIGH` Signal an einen I/O Pin gelegt, darf die dort angeschlossene LED Schaltung nicht mehr als 20 mA "ziehen".
- Wird eine LED Schaltung so verdrahtet, dass die BASIC Stamp die LED mit einem `LOW` Befehl einschaltet, können 25 mA durch den I/O Pin fließen.
- Über P0 bis P7 dürfen zusammen 20 mA fließen. Gleiches gilt für P8 bis P15. Werden mehrere LED Schaltungen angesprochen, müssen höhere Vorwiderstände eingesetzt werden, damit die Stromstärke nicht zu groß wird.

Wenn man in der Lage ist, die Stromstärke in einer Schaltung zu berechnen, kann man auch die Leuchtstärke der LEDs bestimmen. Für jedes elektronische Bauteil gibt es Regeln, wie es auf Spannung, Strom und Widerstand reagiert. Bei einer LED ist die Durchlassspannung die entscheidende Größe, bei einem Widerstand das Ohmsche Gesetz. Wir müssen nun herausfinden, wie viel Strom eine LED Schaltung benötigt. Dabei helfen uns die sogenannten Kirchhoffschen Regeln, die etwas über das Verhalten der einzelnen Ströme und Spannungen in einer Schaltung aussagen.

Vdd – Vss = 5 V Die Spannung zwischen Vdd und Vss beträgt 5 V. Dies ist eine stabilisierte Spannung, die vergleichbar mit einer Batteriespannung von genau 5 Volt ist.

Vin – Vss = 9 V Wird eine 9 Volt Batterie eingesetzt, beträgt die Spannung zwischen Vin und Vss genau 9 V. Aber Vorsicht. Wird über die Steckerverbindung mit einem Steckernetzteil gearbeitet, kann die Spannung, auch wenn 9 Volt eingestellt sind, bis auf 18 Volt hoch gehen.

Erdung und/oder Masse bezieht sich auf den negativen Anschluss der Schaltung. Bei der BASIC Stamp und dem Board of Education, liegt die Masse an Vss. Dort liegen 0 Volt an und bei Einsatz einer 9 Volt Batterie entspricht es dem negative Pol. Der positive Anschluss der Batterie weist 9 Volt auf. Vdd beträgt 5 V (bezüglich Vss), sie wird durch einen Spannungsregler erzeugt, um die BASIC Stamp zu versorgen.

Ohmsches Gesetz: U ~ I Verhält sich in einer Schaltung die Spannung U proportional zur Stromstärke I, spricht man vom Ohmschen Gesetz. Der Proportionalitätsfaktor aus dem Quotienten aus Spannung und Stromstärke entspricht dem Widerstand R.

Durchlassspannung einer Diode : Die an einer durchgeschalteten leuchtenden LED anliegende Spannung zwischen Kathode und Anode beträgt - unabhängig von der Anodenstromstärke durch die Diode - ungefähr 1,6 Volt.

Kirchhoffsche Maschenregel(vereinfacht): Die Summe der an den einzelnen Bauteilen abfallenden Teilspannungen entspricht der anliegenden Gesamtspannung.. Bei einer Schaltung entspricht die Summe der über jedem Bauteil abfallenden Teilspannung der anliegenden Gesamtspannung.

Kirchhoffsche Knotenregel(vereinfacht): Die in einen Knoten hineinfließenden Ströme entsprechen in der Summe den aus dem Knoten herausfließenden Strömen . Die Stromstärke, die über Vdd in eine LED Schaltung hineinfließt, entspricht der Stromstärke, die über Vss abfließt. Ebenso gilt: werden drei LEDs mit einer BASIC Stamp verbunden, wobei jede LED 5 mA benötigt, muss die BASIC Stamp insgesamt 15 mA bereitstellen.

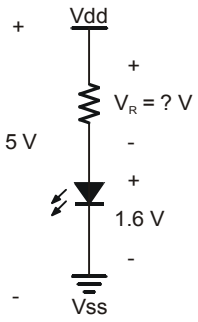
Berechnungsbeispiel: Eine Schaltung, zwei Schaltungen

In zwei Schritten können wir die Stromstärke berechnen, die eine LED Schaltung benötigt:

1. Bestimme den Spannungsabfall über dem Widerstand
2. Berechne mit Hilfe des Ohmschen Gesetzes die Stromstärke durch den Widerstand.

Abbildung F-2 zeigt, wie man den Spannungsabfall über dem Widerstand bestimmt. Zwischen Vdd und Vss liegen 5 Volt an (linke Seite). Die Spannungsabfälle über den Bauteilen werden rechts dargestellt. Unbekannt ist der Spannungsabfall U_R über dem Widerstand R. Über der LED fallen 1,6 Volt ab (Durchlassspannung einer Diode). Nach der Kirchhoffschen Maschenregel wissen wir, dass sich die Summe alle Teilspannungen

auf 5 Volt addiert. Die Differenz zwischen 5 Volt (anliegende Betriebsspannung) und 1,6 Volt (Durchlassspannung Diode) beträgt 3,4 Volt; es ist die über dem Widerstand R abfallende Spannung U_R.



$$V_R + 1.6V = 5V$$
$$V_R = 5V - 1.6V$$
$$V_R = 3.4V$$

Abbildung F-2 Gesamtspannung und Teilspannungen über Widerstand und LED

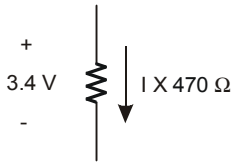
Kilo ist die Abkürzung für 1000. Die Zahl 1000 wird als Kilo bezeichnet und mit dem Buchstaben k abgekürzt. Statt 1000 Ω schreibt man kürzer 1 kΩ: gesprochen: 1 Kiloohm. Entsprechend werden 2000 Ω als 2 kΩ geschrieben.

Milli bedeutet 1/1000 und wird mit einem kleinen m abgekürzt. Fließt in einer BASIC Stamp Schaltung durch eine LED ein Strom von der Stärke 3,4 tausendstel eines Ampere, schreiben wir dafür kürzer: 3,4 mA (3,4 Milliampere).

Was bedeutet mA? Ein Milliampere entspricht einem eintausendstel eines Ampere. Das „m" steht für „Milli", was soviel wie ein tausendstel bedeutet. Das „A" steht für die Einheit Ampere.

Was bedeutet 7.23 mA? Es handelt sich um die durch die LED fließende Stromstärke in Abbildung F-3. Wird der 470 Ω Widerstand gegen einen 220 Ω Widerstand ausgetauscht, fließen durch die Schaltung 15.5 mA und die LED leuchtet heller. Bei einem 1000 Ω Widerstand fließen durch die Schaltung nur noch 3.4 mA und die LED leuchtet nur noch schwach. Entsprechend wird ein 2000 Ω Widerstand die Stromstärke durch die LED weiter herabsetzen (1.7 mA) mit der Folge, dass die LED kaum noch leuchtet.

Abbildung F-3 zeigt ein Beispiel zur Berechnung der Stromstärke in einer Schaltung mit einem 470 Ω Widerstand. Beginne mit dem Ohmschen Gesetz. Bei einer Teilspannung von U = 3.4 V und R = 470 Ω lässt sich die Stromstärke berechnen.

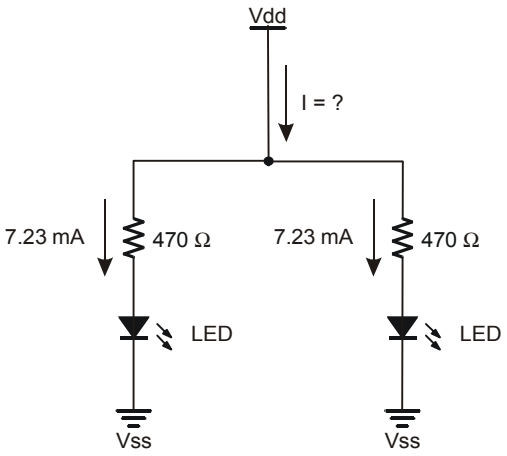


$$V = I \times R$$
$$3.4V = I \times 470\Omega$$
$$I = \frac{3.4V}{470\Omega}$$
$$I = 0.00723 \, V/\Omega$$
$$I = 0.00723A$$
$$I = \frac{7.23}{1000}A$$
$$I = 7.23mA$$

Abbildung F-3
Stromstärke durch den Widerstand

Ja, es ist wahr ! 1 A = 1 V/Ω (Ein Ampere entspricht 1 Volt durch 1 Ohm).

Werden beispielsweise zwei LEDs gleichzeitig eingeschaltet, hat die BASIC Stamp die Schaltung in Abbildung F-4 zu versorgen. Wir müssen als erstes abklären, ob die Stromstärke die Grenze von 20 mA überschreitet. Die Kirchhoffsche Knotenregel besagt, dass der von der Schaltung zur Verfügung gestellte Gesamtstrom der Summe der Teilstromstärken entspricht. Das heißt, dass die Gesamtstromstärke I in Abbildung F-4 der Summe der beiden Teilströme entspricht. Die Summe der beiden Teilströme ergibt 14,5 mA und liegt damit deutlich unter 20 mA. Die Schaltung ist einsetzbar.



$$I = I_1 + I_2 + ...I_i$$
$$I = 7.23mA + 7.23mA$$
$$I = 14.5mA$$

Abbildung F-4
Gesamtstromstärke in einer Schaltung mit zwei LEDs

Du bist dran – Die Schaltung verändern

- √ Wiederhole die Übung in Abbildung F-2, benutze jetzt aber Uin – Uss = 9V. Die Antwort lautet U_R = 7.4 V
- √ Wiederhole die Übung in Abbildung F-3 mit einem Widerstand von 1 kΩ. Antwort: I = 3.4 mA.
- √ Gehe von einem Teilspannungswert von U_R = 7.4 V über dem 1 kΩ Widerstand in Abbildung F-3 aus. Antwort: I = 7.4 mA.
- √ Wiederhole die Übung der Abbildung F-4 mit einem Widerstand von 470 Ω und einem zweiten von 1 kΩ. Antwort: I = 7.23 mA + 3.4 mA = 10.63 mA.

Anhang G: RTTTL Formatanweisungen

Diese Übersicht soll dazu beitragen, das RTTTL Format besser zu verstehen. Die komplette RTTTL Spezifikation ist über Suchmaschinen und dem Suchwort RTTTL auf verschiedenen Sites im Internet einsehbar.

Es folgt ein Beispiel für einen RTTTL formatierten Klingelton:

```
TakeMeOutToTheBallgame:d=4,o=7,b=225:2c6,c,a6,g6,e6,2g.6,2d6,p,
2c6,c,a6,g6,e6,2g.6,g6,p,p,a6,g#6,a6,e6,f6,g6,a6,p,f6,2d6,p,2a6
,a6,a6,b6,c, d,b6,a6,g6
```

Der Text vor dem ersten Doppelpunkt gibt den Namen den Songs wieder, wie er auf dem Display des Handys erscheint. Er lautet in diesem Fall:

```
TakeMeOutToTheBallGame:
```

Zwischen dem ersten und zweiten Komma werden die Song-Einstellungen über d, o und b eingegeben. Es bedeuten:

```
d – Tonlänge
o – Oktave
b – Schläge pro Minute oder Tempo.
```

Der Klingelton TakeMeOutToTheBallGame hat die Einstellungen:

```
d=4,o=7,b=225:
```

Die Melodienoten werden, durch Komma getrennt, nach dem zweiten Doppelpunkt eingegeben. Wird nur der Notenname eingegeben, wird die Note in der voreingestellten Tonlänge und Oktave wiedergegeben. Die zweite Note in TakeMeOutToTheBallGame ist zum Beispiel:

```
,c,
```

Da keine weiteren Informationen vorliegen, wird sie als Viertelnote (d=4) in der siebten Oktave (o=7) gespielt

Eine Note kann mit bis zu fünf Zeichen beschrieben werden; ihre Bedeutungen zeigt das folgende Schema:

```
,Tonlänge  Note  Halbton auf/ab  Punktiert  Oktave,
```

Ein Beispiel:

```
,2g#.6,
```

bedeutet, die punktierte halbe Note gis wird in der sechsten Oktave gespielt.

Es folgen weitere Beispiele aus TakeMeOutToTheBallGame:

```
,2g.6, – Halbe Note, G, punktiert, 6. Oktave
,a6,   – Viertelnote, A, 6. Oktave
,g#6,  – Viertelnote, G#, 6. Oktave
```

Das Zeichen:

```
,p,
```

markiert eine Pause. Ohne Zusatzinformation wird eine Pause von einer Viertelnotenlänge „gespielt". Es lassen sich aber auch Pausen von einer halben Notenlänge eingeben:

```
,2p,
```

Ein Beispiel für eine punktierte Pause:

```
,2p.,
```

In diesem Fall entspricht die Pause einer halben plus einer Viertelnotenlänge.

Index

- % -

%, 187

- * -

*, 99
**, 262
*/, 99

- 3 -

3-fach Schiebeschalter, 17

- 7 -

7-Segment Anzeige, 175–81

- A -

Abfrage, 94
AD5220 Digitalpotentiometer
  Pinbelegung, 287
  Pinbezeichnungen und Funktionen, 287
Aktiv-HIGH, 83
Aktiv-LOW, 83
Algorithmus, 100
American Standard Code for Information Interchange, 31
Ampere, 341
Anode, 43
Anschluss
  Piezo Schallwandler, 236
ASCII, 31, 271

- B -

BASIC Stamp
  BASIC Stamp 2 Modul, 1
  HomeWork Board, 327
  Komponenten, 329
BASIC Stamp Editor
  Identify, 338, 339
  Problembehandlung, 337
  Software, 6
  Speicherbelegung, 281
BASIC Stamp HomeWork Board, 14
BASIC Stamp HomeWork Board
  Komponenten, 331
Basis, 284
Batterie, 46
Bauteilzeichnung
  7-Segment Anzeige, 175
  DUO LED, 63
  Fotowiderstand, 202
  LED, 43
  Piezo Schallwandler, 235
  Potentiometer, 148
  Taster – einfacher Schließer, 75
  Transistor (NPN), 283
  Widerstand, 41
binäre Zahlen, 21, 81, 187
Bit, 56, 265
Board of Education, 13
  Full Kit, 325
  Komponenten, 330
  Servo Wannenstecker, 112
  Version, 111

Board of Education Rev B
  Komponenten, 332
Boe-Bot, 3
Byte, 56, 265
- C -
Cadmiumsulfid, 201
COM Port, 16, 209
Compilerdirektiven, 22, 25
CR, 26
- D -
DATA, 247
DEBUG
  Befehl, 26
DEBUG Formatanweisung
  CR, 26
  DEC, 26
Debug Terminal, 24
  Fenster Empfangen, 127
  Fenster Senden, 127
DEBUGIN, 126–30, 312
DEBUGIN Formatanweisungen
  STR, 312
DEC, 26, 127
Digitalpotentiometer, 286–95
DIRH, 185–88
DUO LED, 63
Durchlassspannung einer Diode, 345
- E -
Echo, 127
EEPROM, 210–16, 247, 281
Einbindung von Subsystem, 301
Eingabe/Ausgabe Pins, 44
Eingebettetes System, 1
Einsprungmarke, 219
Elektron, 47, 341
Emitter, 284
END, 26
Erdung, 345
EXIT, 313
- F -
Feld, 312
Fenster Empfangen, 127
Fenster Senden, 127
Fernsteuerung, 162
FOR…NEXT, 55–58, 120
Fotowiderstand, 201–2
  RC-Schaltung, 203
FREQOUT, 237–38
Frequenz
  Mischen, 238, 242–44
  Musiknoten, 244–46
  Tondauer, 239
- G -
ganze Zahl, 262
Ganzton, 244
Garantie, 3
geschachtelte Schleife, 240–42
GOSUB, 219
- H -
Halbton Erhöhung, 244
Halbton Erniedrigung, 244
Hertz, 235
Hervorhebung der Syntax, 25

HIGH, 51
Holen und Ausführen, 281
Hz. siehe Hertz

- I -

I/O Pins. siehe Eingabe/Ausgabe Pins
IC. siehe Integrierter Schaltkreis
Identification Fenster, 20
Identify, 338, 339
IF…ELSEIF…ELSE, 89
IF…THEN…ELSE, 85
Impuls, 119
IN3, 81
IN7, 160
Input/Output Pin
  Input, 75
Input/Output Pins
  Voreinstellung, 187
Integrierte Schaltung
  Pinbelegung, 286
  Referenzkerbe, 286
Integrierter Schaltkreis, 282
Interferenz, 243

- J -

Jumper, 112

- K -

Kathode, 43
kB, 211
KB, 211
KCL, 345
Kilo, 346
Kilobyte, 211
Kirchhoffsche Gesetze (vereinfacht)
  Knotenregel, 345
  Maschenregel, 345
Klaviertastatur, 244
Klingelton, 263
Kollektor, 284
Kommentar, 25, 54
Komponenten
  BASIC Stamp, 329
  BASIC Stamp HomeWork Board, 331
  Board of Education, 330
  Board of Education Rev B, 332
Kondensator, 151–54, 158
  Anschlussbelegung, 151
  keramisch, 159
Kontrollleuchte, 39
KVL, 345

- L -

Ladung, 341
Leiter, 341
Light Emitting Diode, 39–40, 42–43
  Anode, 43
  Anschlüsse, 43
  DUO, 63
  Kathode, 43
  Schaltsymbol, 43
  Schaltungsberechnung, 345
LOOKDOWN, 192
LOOKUP, 189–91
LOW, 51

- M -

mA, 346
Masse, 345
Mathematische Operationen, 260
Menü, 307
Metrisches System, 346
Mikrocontroller, 1
Mikrosekunde, 120
Milli, 346
Millisekunde, 51, 120
Mischen, 238, 242–44
Multiplikation, 99
Musik
  FREQOUT und Noten, 245
  Ganzton, 244
  Halbton Erhöhung, 244
  Halbton Erniedrigung, 244
  Klaviertastatur, 244
  Klingelton, 263
  Noten, 244
  Notenlänge, 246, 248, 250–55
  Pause, 250
  Punktiert, 255
  RTTTL Format, 349–50
  RTTTL Format, 263–64, 269–75
  Tempo, 250

- N -

nc, 177
neutral, 47
Nib, 56, 265
No-Connect, 177
Noten, 244
Notenlänge, 248, 250–55
NPN, 283

- O -

Offset, 164
Ohm, 40, 341
Ohmsches Gesetz, 345
ON…GOSUB, 307–17
Operator, 99
OUTH \r, 184

- P -

Parallax Standard Servo, 109
  Bauteile, 110
  Hinweis, 110
  Hinweis zur Spannungsversorgung, 335
  Warnung, 163
Parallel
  Bus, 183
Parallel Device, 183
Pause, 250
PAUSE, 51
PBASIC Befehle
  DEBUG, 26
  DEBUGIN, 126–30, 312
  DEC, 127
  DO…LOOP, 51
  END, 26
  EXIT, 313
  FOR…NEXT, 55–58, 120
  FREQOUT, 237–38

GOSUB, 219
HIGH, 51
IF…ELSEIF…ELSE, 89
IF…THEN…ELSE, 85
LOOKDOWN, 192
LOOKUP, 189–91
LOW, 51
ON…GOSUB, 307
PAUSE, 51
PULSOUT, 119
RANDOM, 99
RCTIME, 160–62, 161
READ, 215–16, 247
RETURN, 220
SELECT…CASE, 264–69
TOGGLE, 291
UNTIL, 97
WRITE, 212–14, 247

PBASIC Direktive
DATA, 247–50
PBASIC, 25
PIN, 307
Stamp, 25

PBASIC I/O Register
DIRH, 185–88
IN3, 81
OUTH, 185–88

PBASIC Operatoren
%, 187
*, 99
**, 262
*/, 99
Bearbeitungsregeln, 260
DCD, 261
Klammern, 260

Piezo
Element, 237
Schallwandler, 235

Pinbelegung, 176, 286
Potentiometer, 147–49
Anschlüsse, 148
Digital, 286–95

Prototypfeld
Buchsenleiste, 44
Eingabe/Ausgabe Pins, 44

Pseudozufall, 100
Pull-Up Widerstand, 83
PULSOUT, 119
Punktierte Noten, 255

- R -

RANDOM, 99
RCTIME, 160–62, 161
READ, 212, 247
Referenzkerbe, 286
RETURN, 220
Roboter, 2–4
RTTTL Format, 263–64, 269–75, 349–50

- S -

Schallwandler, piezoelektrisch, 237
Schaltkreis, 44
Schaltplan
  Zeichnung, 47
Schaltsymbol
  7-Segment Anzeige, 176
  DUO LED, 63
  Fotowiderstand, 202
  LED, 43
  Piezo Schallwandler, 235
  Potentiometer, 148
  Schaltungspunkte stehen für elektrische Verbindungen, 88
  Taster – einfacher Schließer, 75
  Transistor (NPN), 283
  Widerstand, 40
Schwebung, 243
Schwingung, 235
SELECT…CASE, 264
Sensor, 201
Sensorfeld, 301, 307
Servo, 109–11
  Anschlussbuchse, 109
  Gehäuse, 110
  Hinweis zur Spannungsversorgung, 335
  Kabel, 109
  Potentiometersteuerung, 163
  Steuerhorn, 109
  Verbindung mit der BASIC Stamp, 111–19
  Wannenstecker, 112
  Warnhinweis, 110, 163
Skalierung, 164
Spannung, 47, 341
Speicherbelegung, 210, 281
Stamp Plot Lite, 205–10
Startwert, 100
Statusanzeige, 39
Steckbrett. siehe Prototypfeld
STR, 312
Strom, 40, 46, 47, 341
  Fluss, 47
  Milliampere, 346
Stromrichtung
  physikalisch, 47
  technisch, 47
Superposition, 243

- T -

Taster, 75–76
  einfacher Schließer, 76
  Schaltung, 79–81
Tempo, 250
TOGGLE, 291
Token, 281
Toleranz, 41
Tondauer, 239
Tonhöhe, 235
Tonlänge, 246
Transistor, 282, 283–86
  2N3904, 284

Basis, 284
Basisstrom, 284
Emitter, 284
Kollektor, 284
NPN, 283
Schalter, 285

- U -

Überlauf, 265
Unterbrechen der Spannungsversorgung, 33
Unterprogramm, 219–27
Einsprungmarke, 219

UNTIL, 97
US232B, 323
USB-Seriell Adapter, 6, 16, 323

- V -

Variable, 55, 56, 265
Bit, 56, 265
Byte, 56, 265
Feld, 312
Nib, 56, 265
Word, 56, 265

Vdd, 44, 48, 345
Vin, 44, 345
Volt, 341
Vss, 44, 48, 345

- W -

What's a Microcontroller
Buch, 325
Parts Kit, 325

Widerstand, 40–42, 50, 341
Anschlüsse, 40
Farbcodes, 41–42
Pull-Down, 83
Pull-Up, 83
Toleranz, 41

Word, 56, 265
WRITE, 212–14

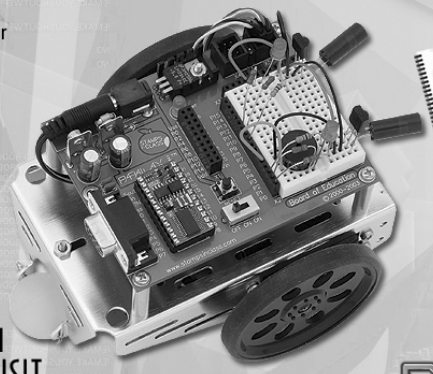
- Z -

Zählvorgang, 94
Zeilenrücklauf, 26

Learn Robotics with the Boe-Bot

If you enjoyed learning microcontroller programming with the BASIC Stamp, why not continue the learning curve by building a robot?

The *Robotics!* curriculum uses your BASIC Stamp 2 module and Board of Education to create a rolling robot that can follow or avoid light, detect and avoid objects with infrared, and much more.

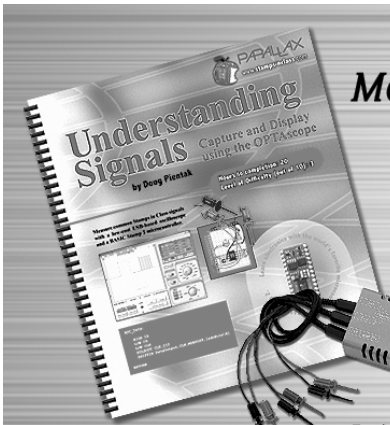


Robotics!
Parts & Text
#28154

For more information or to order online visit www.parallax.com/sic

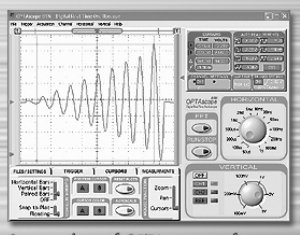
PARALLAX

Measure and Interpret Signals



OPTAscope USB oscilloscope with Understanding Signals text and components (order #28119). This new scope is available exclusively through Parallax and includes complete BASIC Stamp support. Learn how to view, measure and generate signals used for servo control, sine waves, op-amp buffers, voltage amplifiers, infrared emitters and synchronous and asynchronous serial communication.

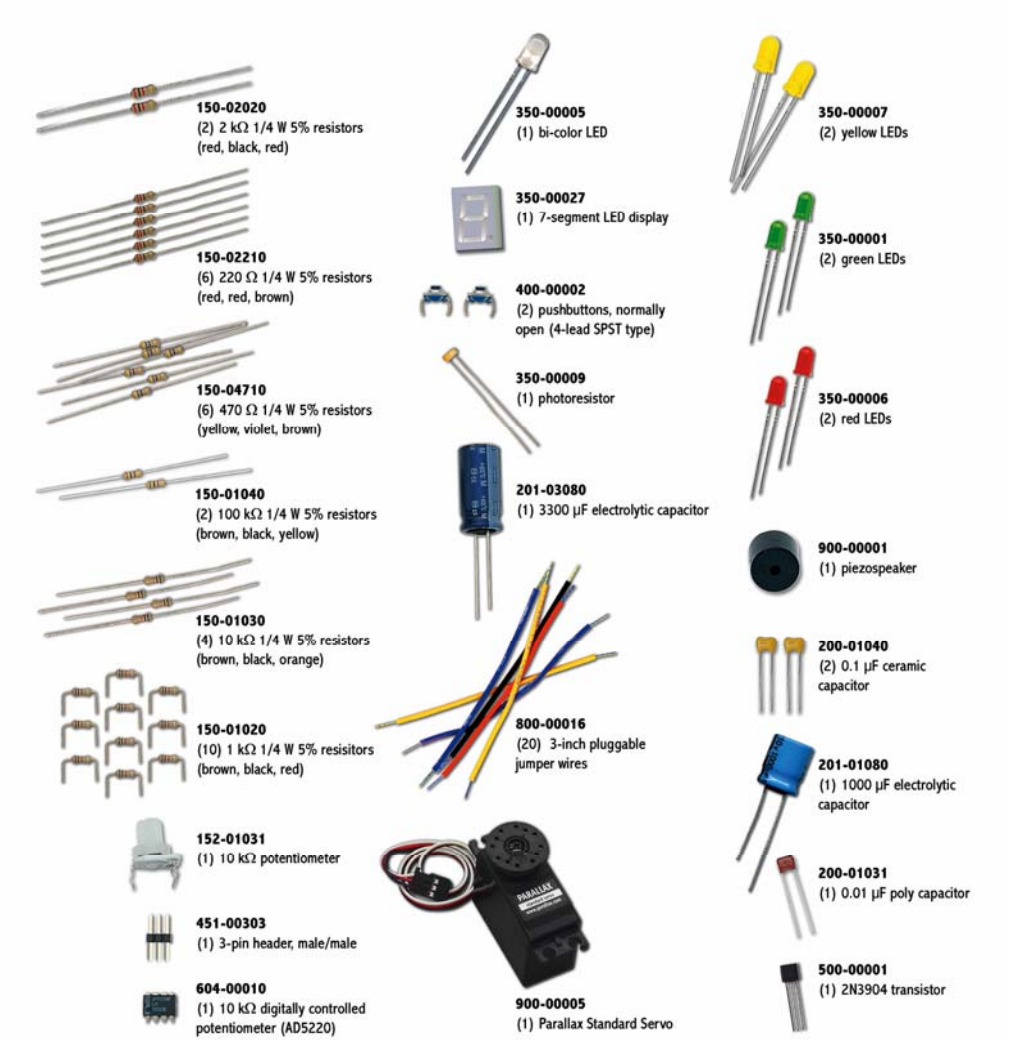
Order #28119



Screen shot of OPTAscope software.

PARALLAX

www.parallax.com/sic



Bauteile und Mengenangaben der verschiedenen Bausätze der *What's a Microcontroller kits* unterliegen ständigen Veränderungen. Die Bauteile müssen nicht mit den abgebildeten übereinstimmen. Bei Fragen zu Ihrem Bausatz wenden Sie sich bitte an stampsinclass@parallax.com.